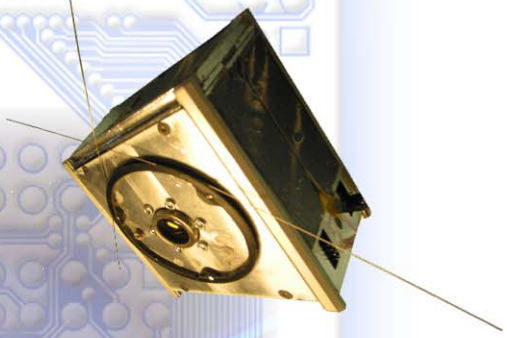
Camera System
for Pico Satellites



E4 - Spring 2006
06gr415

Aalborg University

Henrik Dalsager

Thomas Esbensen

Michael Niss

Christoffer Sloth

Steffen Villadsen

AALBORG UNIVERSITY

Faculty of Engineering and Science
Institute of electronic systems
Fredrik Bajers Vej 7 A1
9220 Aalborg Ø
Denmark
http://ies.aau.dk

Title:
Camera System for Pico Satellites

Subject:
Microcomputer System

Project period:
E4 2006

Project group:
06gr415

Members of the project group:
Henrik Dalsager
Thomas Esbensen
Michael Niss
Christoffer Sloth
Steffen Villadsen

Supervisor:
Karl Kaas Laursen

Number printed:
8

Number of pages:
Report: 134
Total: 199

Ended:
May 29th 2006

Abstract:

As a feasibility study for the future AAUSAT-III satellite, it is investigated which requirements should be taken into consideration when designing a camera system for a pico satellite traveling in Low Earth Orbit. The requirements lead to the design of a camera subsystem that provides an interface between an onboard computer and the chosen image sensor.

The image sensor used in the project is supplied as a stabilized module by Devitech ApS, and outputs parallel data. To achieve the necessary speed for collecting the data a FPGA is inserted to act as a DMA controller.

The software controlling the camera system is based on eCos and is designed to compress, save, delete, produce thumbnails of, and list up to five images captured. It is also possible to store one raw image to make it optional if the image processing happens on the satellite or on Earth.

The camera system communicates with the rest of the satellite using the existing protocol that is implemented on AAUSAT-II. Furthermore, the software of the camera system can be updated using the RS-232 debug interface.

The PCB design of the prototype attempts to have EMC and follows ESA space graded standards; the PCB is manufactured by GPV Printca A/S.

The camera system was successfully tested by fetching test data from the image sensor; yet no image with a recognizable subject is captured due to lack of time for adjusting the lens system used for testing.

AALBORG UNIVERSITET

Det Teknisk-Naturvidenskabelige Fakultet
Institut for elektroniske systemer
Fredrik Bajers Vej 7 A1
9220 Aalborg Ø
Danmark
http://ies.aau.dk

Titel:
Kamerasystem til Pico-satelliter

Tema:
Mikrocomputer System

Projektperiode:
E4 2006

Projektgruppe:
06gr415

Projektgruppens medlemmer:
Henrik Dalsager
Thomas Esbensen
Michael Niss
Christoffer Sloth
Steffen Villadsen

Vejleder:
Karl Kaas Laursen

Oplag:
8

Sideantal:
Rapport: 134
Samlet: 199

Afsluttet:
29. maj 2006

Synopsis:

Som forundersøgelse til den kommende AAUSAT-III satellit er det undersøgt, hvilke krav, der skal stilles til et kamerasystem beregnet for en pico-satellit i kredsløb om Jorden i Low Earth Orbit. Kravene leder direkte frem til design af et kamerasystem, der stiller en grænseflade til rådighed til kommunikation mellem on-board computeren og den anvendte billedsensor.

Den benyttede billedsensor er leveret af Devitech ApS og sender billeddataene ud parallelt. Til håndtering af de modtagne data med en tilstrækkelig hastighed er en FPGA benyttet som DMA controller.

Kamerasystemets software er baseret på operativsystemet eCos, og er designet så det kan komprimere, gemme, slette, producere thumbnails af, og levere en liste af op til fem billeder. Det er desuden muligt at gemme rå billeddata, således det kan vælges om billedbehandlingen skal ske i satellitten eller på jorden.

Kamerasystemet kommunikerer med resten af satellitten ved hjælp af den eksisterende protokol fra AAUSAT-II. Foruden dette, er det muligt at opdatere og debugge softwaren ved hjælp af en RS-232 port.

Printudlægget til prototypen er forsøgt designet, således det er EMC-rigtigt og følger ESAs rumkvalifikationsstandarder. Prototypeprintet er fremstillet af GPV Printca A/S.

Kamerasystemet kan hente et testmønster fra billedsensoren, men pga. manglende tid til indstilling af linse og billedsensor, blev der ikke taget et genkendeligt billede.

Preface

This report provides documentation about the development of the digital camera system for the AAUSAT-III satellite. The camera system is developed by project group 06gr415 on fourth semester on Aalborg University. It is based on components suitable to the environment in space and the software is prepared to operate within the frames of a pico satellite.

The report sets up requirements for the camera system, describes how it is designed, how to interface with it, and preliminary system tests are described. The project is limited by the time span of a single semester and even though a prototype has been constructed, some further development is needed before being ready to flight; this is described in the perspectives found after the conclusion.

The intended readers for the report are students interested in space- and, or camera technology. Though it is especially intended for students involved with the work on AAUSAT-III. The project was initiated as a feasibility study to establish the possibilities of building a digital camera for AAUSAT-III. The mission for AAUSAT-III is not yet established and as a consequence, effort was made to design the camera system after the specifications provided to subsystems on AAUSAT-II.

The goals for this report are to:

- Establish a basic knowledge of the specific problems that occurs when designing electronics for space environments.
- Provide a general understanding of lenses and digital camera technology.
- Introduce the considered hardware possibilities and document the choices of them.
- Show how a "use case model" can be utilized to analyze the functionalities that must be provided for the user of the camera system.
- Combine the research into a requirement specification that the camera system must fulfill before launch.
- Explain through documentation the choice of components.
- Document the hardware design that was evolved from the requirement specifications; both the layout of connections as well as the physical implementation of the prototype.
- Show how an embedded software can be designed using a combination of development tools when basing the design on an embedded operating system.
- Give an insight into the tools used for describing the design, i.e. flowcharts based on UML active state chart figures.
- Present the tools used for development and debugging.
- Document and describe the use of image manipulation methods.
- Expose the secrets of interpolation and image compression and document the implemented algorithms.
- Evaluate the prototype camera system in a conclusion and describe future improvements in a section describing the perspectives of the project.

The report is written in American English, and a synopsis in Danish is attached.

A word abbreviation is found following the preface, giving an overview of the technical terms used. Some of these terms are specifically related to the existing satellite platforms and cannot be assumed common knowledge to all of the intended readers. It is recommended for the reader to skip the word abbreviation and use it as reference when an unfamiliar expression is encountered.

Appendixes are found after the perspectives, and an index is found last in the report. For easy use of the camera prototype during AAUSAT-III development, a users manual is attached, providing jumper settings and other basic information about the interface.

On page 199 a CD is found containing additional documentation. This also includes code documentation as HTML files created using DoxyGen.

In the development process contacts were made to the companies Devitech ApS and GPV Printca A/S.

Devitech ApS was contacted to gain access to the image sensor market and give a selection of image sensors to choose from. "Devitech specialises in development and supplies of advanced embedded computer vision systems and digital camera products" [Devitech, 2004] and in this project they supplied not only image sensor, but also a prototype for their own BlackEye platform which is designed to stabilize the image sensor; documentation of the circuit was included. Devitech ApS also provided a lens for testing the equipment that can be mounted on the BlackEye platform. However, a non disclosure agreement was formed and agreed upon, limiting the possibilities of documenting their design, but allowing the project group to implement it.

GPV Printca A/S was contacted during the development phase as it was realized that the components needed for space implementation was SMD components and that the possibilities for designing the necessary PCB was limited on AAU. GPV Printca A/S agreed to sponsor the production of PCBs to the satellite program on AAU provided that the designs were ESA space graded compliant. However, this deal did not provide access to free PCBs and to stay within budget of the project the AAUSAT-II team was contacted to help out financially. The contract that was formed included production of the AAUSAT-II engineering and flight models on the same panel as the camera system prototype. To help meeting the design specifications, GPV Printca A/S provided design guidelines regarding PCB layout, and these were implemented in the design of the PCB.

We especially thank Associate Professor Ole Olsen who has provided help in designing VHDL code for the chosen FPGA.

We also thank GPV Printca A/S and Devitech ApS for sponsoring the project and thus making it possible to realize the design.

We thank the AAUSAT-II developers for CVS access to preliminary documentation and for lending us a prototype of the OBC for software testing.

Definitions in the Report

A kilobyte is defined as 1024 bytes and is written as kB.
A kilobit is defined as 1024 bits and is written as kb.
A megabyte is defined as 1024 kilobytes and is written as MB.
A megabit is defined as 1024 kilobits and is written as Mb.

A byte is defined as 8 bits.
A half word is defined as 16 bits.
A word is defined as 32 bits.
A Mpixel is defined as 1,048,576 pixels.

Word Abbreviation

The explanations below here hold for AAUSAT-II, but similar contents are expected for AAUSAT-III. This section should be regarded as a work of reference.

- **ACS - Attitude Control System**
  The control part of ADCS. See ADCS.

- **ADCS - Attitude Determination and Control System**
  ADCS consists of a determination and a control system; the control part of ADCS is ACS. If the satellite is tumbling, ADCS can determine how much and in which direction. ADCS can also detumble the satellite by using magnetorquers and momentum wheels. ADCS also conducts fine pointing for e.g. photographing purposes.

- **ADS - Attitude Determination System**
  The determination part of ADCS. See ADCS.

- **API - Application Programmable Interface**
  The API is the documentation that programmers can fetch about a system. It informs the programmer about which functions the system allows for the programmer to use. In operating systems this could be functions provided by the OS that will allow the programmer to access different types of hardware in a similar manner.

- **ASIC - Application Specific Integrated Circuit**
  An ASIC is an integrated circuit (IC) customized for a particular use rather than intended for general-purpose use [Wikipedia, 2006a].

- **Beacon**
  The satellite transmits data containing time and housekeeping with a certain interval allowing MCC to track the satellite and get information about its state.

- **BGA - Ball Grid Array**
  BGA components are contained in a package with virtually no pins. Instead small balls of metal connects the pads to the IC core. The basic idea is that connections can be made underneath the housing, and therefore many connections can be added to a chip without increasing the surface area [Coorporation, 2003].

- **CAN - Controller Area Network**
  CAN is a communication bus that is used in a variety of designs. The bus specifications are defined in the ISO 11898/11519 standard. It is a balanced (differential) 2-wire interface running with a "Non Return to Zero" encoding much like RS-232.

- **CCD - Charge Coupled Device**
  "A charge-coupled device (CCD) is an image sensor, consisting of an integrated circuit containing an array of linked, or coupled, capacitors sensitive to the light. Under the control of an external circuit, each capacitor can transfer its electric charge to one or other of its neighbors. CCDs are used in digital photography and astronomy (particularly in photometry, optical and UV spectroscopy and high speed techniques such as lucky imaging)." [Wikipedia, 2006c]

- **CDH - Command and Data Handler**
  CDH is software on the OBC that verifies and interprets all the internal and external commands. CDH takes care of all heavy calculations and keep track of everything in the satellite. If a unit inside the satellite needs to log something, this is also handled by CDH. Contact to Earth except basic packages from COM also has to go through CDH. CDH is running multiple program threads for all of the subunits.

- **Chunk**
  A chunk is a data packet containing a part of the entire data set.

- **COM - Communication System**
  COM is the communication system to the outside world. All communication to and from the satellite has to go through the COM subsystem, which is the space segment of the spacelink.

- **CPLD - Complex Programmable Logic Device**
  A device consisting of several PLDs and internal interconnects between them. See PLD.

- **EBI - External Bus Interface**
  The EBI generates the signals to access the external memory and peripheral devices and is programmed through the Advanced Memory Controller of the ARM microprocessor.

- **eCos - embedded Configurable operating system**
  eCos is an embedded operating system that provides a uniform interface to the applications. Systems using eCos are easy to update, since the application can be coded in a way where they are not processor specific for most of the software.

- **EMC - Electro Magnetic Compatibility**
  To avoid emitting or receiving noise, thus causing damage to other products, or making one self unable to function, Electro Magnetic Compatibility must be ensured. Problems with EMC occurs not only when connected to other circuits, but also when being close to them as electro magnetic waves can be transferred into circuits. This is especially the case for high frequency circuits.

- **EPS - Electrical Power Supply**
  The power supply delivers power to all the units in the satellite and is capable of shutting down the units separately. EPS keeps track of the battery voltage and if this is running dangerously low, all non important units are shut down. The solar cells of the satellite are also managed by EPS.

- **ESA - European Space Agency**
  The European pendent to NASA. “The European Space Agency is Europe’s gateway to space. Its mission is to shape the development of Europe’s space capability and ensure that investment in space continues to deliver benefits to the citizens of Europe.” [ESA, 2006]

- **FLP - Flight Planner**
  FLP is one of the threads running on OBC. This thread is basically a calendar, which handles the execution of specific commands at specific times. Events in FLP can be added and removed from Earth.

- **FPGA - Field Programmable Gate Array**
  A FPGA is a semiconductor device containing programmable logic components, which can be programmed to duplicate the functionality of basic logic gates or more complex combinatorial functions such as decoders or simple math functions. The programming is done by the customer, not by the manufacturer, and the term field refers to the world outside the factory.

- **FR4 - Flame Retardant Type 4**
  “FR4 laminate is the usual base material from which plated-through-hole and multilayer printed circuit boards are constructed. ”FR” means Flame Retardant, and Type ”4” indicates woven glass reinforced epoxy resin.” [EMT, 2006]

- **GND - Ground Station**
  GND is the location on Earth that handles the radio connection to the satellite, making it the ground segment of the spacelink. Every contact from Earth has to go through the ground station. Mainly, GND is an antenna, a radio and some kind of encoding/decoding equipment.

- **HK - Housekeeping**
  Housekeeping is data provided by the subsystems about the state of the satellite. The data is usually temperature and battery levels, but can also contain information about what the subsystems are doing.

- **HSN - Hardcore Satellite Network**
  The protocol for transmitting data on AAUSAT-II is implemented on a CAN bus and is developed especially to the AAU satellites by students. It is an extremely reliable protocol, and its implementation can be done with a low memory usage.

- **I^2C - Inter-Integrated Circuit**
  A serial Bus specification developed by Philips. It is used in a wide variety of commercial products and uses a two-wire unbalanced connection, thereby it is more sensitive to noise than balanced standards like RS-432 and CAN.

- **INSANE - INternal SAtellite NEtwork**
  This was the original protocol for communication on AAUSAT-II. It is in many of the sources in the bibliography described as implemented, but has later been replaced by HSN.

- **JTAG - Joint Test Action Group**
  "While designed for printed circuit boards, it is nowadays primarily used for testing sub-blocks of integrated circuits, and is also useful as a mechanism for debugging embedded systems, providing a convenient "back door" into the system. When used as a debugging tool, an in-circuit emulator which in turn uses JTAG as the transport mechanism enables a programmer to access an on-chip debug module which is integrated into the CPU via JTAG. The debug module enables the programmer to debug the software of an embedded system." [Wikipedia, 2006f]

- **LEO - Low Earth Orbit**
  LEO is a type of orbit close to Earth.

- **MCC - Mission Control Center**
  MCC is the controlling unit on Earth.

- **MECH - Mechanical System**
  MECH is the mechanical structure that keeps the subsystems in place.

- **OBC - On-board Computer**
  OBC is the hardware basis for the CDH software. It is a combination of a CPU, memory and other necessary hardware.

- **OS - Operating System**
  Operating Systems are software that enables programmers to handle different hardware in a uniform way. Best known examples are Windows and Linux that provide a uniform platform for applications, so that they run in a similar fashion without considering the specific hardware.

- **PCB - Printed Circuit Board**
  Electrical insulating material with electrical traces. Modern PCBs feature multi layers with connections between the traces named VIA holes. The surface and sublayers use copper traces to provide electrical connections for chips and other components.

- **PEEL - Programmable Electrically Erasable Logic**
  Similar to Generic Array Logic (GAL) see PLD [Wikipedia, 2006g].

- **PLD - Programmable Logic Device**
  A programmable logic device is a device with a fixed structure of logic gates which can be programmed to emulate different logic circuits by fusing internal connections. PLD is the generic term covering a variety of different technologies. The first type were programmable logic array (PLA) that consisted of an array of AND and OR gates [Wakerly, 2000, p. 15]. Improved functionality such as reprogramability, configurable I/O direction and the ability to emulate flip-flops were introduced in the later types such as GAL devices [Wakerly, 2000, p. 343]. PLDs are typically programmed using the hardware description language ABEL. Due to their structure, PLDs cannot be scaled to larger sizes and instead CPLDs or FPGAs are used [Wakerly, 2000, p. 15].

- **PL - Payload**
  Payload is the part of the satellite that pays for the mission. In other words, the payload is the mission objective.
- **PPOD - Poly Pico satellite Orbital Deployer**
  PPOD is the satellite launching unit, which carries a satellite within a rocket during launch. Later, in orbit, the PPOD releases the satellite from the launch vehicle [University, 2004a].
- **SATlink - Satellite Uplink/Downlink**
  The connection between the GND and the satellite is divided into an uplink and a downlink. The uplink is the part of the connection that is used for transmitting commands and uploading software to the satellite, where the downlink is used by the satellite for beacons and data transfers.
- **SSU - Struktureret System Udvikling**
  SSU is a danish toolbox for systems development. The tools provided are primarily aimed at software development, but is transposed to hardware design also [Bierring-Sørensen et al., 2002].
- **Subsystem**
  A part of the satellite that provides a functionality; examples of subsystems are EPS, ADCS and OBC.
- **Thumb - Thumbnail**
  A thumbnail is a term used in photo related material to describe a miniature copy or low resolution version of an existing image, allowing the viewer to get a quick overview of the images in a library or photo shoot.
- **UML - Unified Modeling Language**
  UML offers tools to design software ensuring that the implementation of code can be done by multiple programmers without extended knowledge of each programmers codes. The UML standards are maintained by the Object Management Group [Lee and Tepfenhart, 2001].

Contents

1 Introduction **16**
1.1 Purpose . . . 16
1.2 General System Description . . . 17

2 Analysis **19**
2.1 Radiation in Space . . . 19
2.1.1 Consequences of Radiation . . . 19
2.1.2 Hardware Protection Against Radiation . . . 20
2.1.3 Software Protection Against Radiation Consequences . . . 21
2.2 Camera Structure . . . 23
2.2.1 Light Intensity of the Subject . . . 23
2.2.2 The Lens System . . . 24
2.2.3 Exposure of a Camera . . . 26
2.3 Image Sensors . . . 28
2.3.1 CCD . . . 28
2.3.2 CMOS . . . 29
2.3.3 Comparison of CCD and CMOS . . . 30
2.3.4 Exposure Reset Operations of Image Sensors . . . 30
2.4 Microcomputer Hardware . . . 32
2.4.1 Hardware Functionality . . . 32
2.5 Software Description . . . 34
2.5.1 Use Cases . . . 34
2.6 Communications Protocol . . . 38
2.6.1 Network Design . . . 38
2.6.2 Packet Handling . . . 39

3 Camera System Requirement Specification **41**
3.1 Satellite Requirements . . . 41
3.2 Functional Requirements . . . 42

4 Test Specifications **44**
4.1 Test Description of Satellite Requirements . . . 44
4.2 Test Description of Functional Requirements . . . 45

5 System Description **48**
5.1 Choice of Components . . . 48
5.1.1 Overall Component Considerations . . . 48
5.1.2 Image Sensor . . . 49
5.1.3 Microcontroller . . . 51
5.1.4 Short Term Storage . . . 53
5.1.5 Long Term Storage . . . 53
5.1.6 Control Logic . . . 53
5.1.7 Summary of Chosen Components . . . 54
5.2 Electric Circuit Design . . . 55
5.2.1 Address and Data Bus . . . 55
5.2.2 Memory Control . . . 56
5.2.3 Image Sensor Control . . . 56
5.2.4 External Connectors . . . 57
5.2.5 Configuration and Control Pins for FPGA . . . 58

5.2.6 Summary of Connections . . . 58
5.3 Realizing the Electric Circuit Design . . . 59
5.3.1 Determination of Component Values . . . 59
5.3.2 PCB Design . . . 60
5.4 Implementation of Logic in FPGA . . . 62
5.4.1 FPGA Chip Architecture . . . 62
5.4.2 Definition of Sequences . . . 63
5.4.3 FPGA Requirements and Premises . . . 64
5.4.4 Assigning PIO Connections . . . 65
5.4.5 Truth Table of Output Values . . . 66
5.4.6 Specifying Read-out Sequence . . . 66
5.4.7 Timing Requirements . . . 67
5.4.8 Programming the FPGA . . . 71
5.4.9 Description of FPGA Block Diagram . . . 72

6 Software Design 75
6.1 Process Overview . . . 75
6.2 Implementation of Processes . . . 76
6.3 Initializing the System . . . 77
6.3.1 Microcontroller Boot Sequence . . . 77
6.4 Memory Usage in the Camera System . . . 80
6.4.1 Layout of Memory in FLASH . . . 80
6.4.2 Image Info Structure in FLASH . . . 83
6.4.3 Image Info Structure in RAM . . . 83
6.4.4 Software in RAM . . . 84
6.5 Software Flow of the Multi Threaded Applications . . . 84
6.5.1 Thread Design . . . 85
6.5.2 Function Design . . . 86

7 Implementation of Subfunctions 91
7.1 Read-out Temperature . . . 91
7.2 Initializing Image Sensor . . . 91
7.2.1 Communication Between ARM and Image Sensor . . . 92
7.3 SRAM Software . . . 93
7.4 Suspending the Image Sensor . . . 94
7.5 Color Interpolation . . . 94
7.5.1 Demosaicing . . . 95
7.5.2 Choice of Interpolation Method . . . 97
7.6 Resizing Image . . . 98
7.7 Image Compression . . . 98
7.7.1 Color Channels of an Image . . . 98
7.7.2 Block Prediction . . . 101
7.7.3 Discrete Cosine Transform and Quantization . . . 101
7.7.4 Comparison and Choice of Image File Format . . . 103

8 Testing 106
8.1 Block Tests of the FPGA . . . 106
8.1.1 Setup Used for Circuit Test/Verification . . . 106
8.1.2 Mode Independent Pass . . . 106
8.1.3 Pass-through Mode . . . 107
8.1.4 Read-out Mode . . . 110
8.2 Module Tests . . . 116
8.2.1 Command Buffer . . . 116
8.2.2 Setup Camera . . . 116
8.2.3 Capture Image . . . 117
8.2.4 List Image . . . 118
8.2.5 Delete Image . . . 119

8.2.6 Send Image . . . 120
8.3 Integration Test . . . 121
8.3.1 Testing Read-out Functionality . . . 121
8.4 Acceptance Test . . . 122
8.4.1 Results of Testing §SAT9 . . . 122
8.4.2 Results of Testing §SAT10 . . . 122
8.4.3 Results of Testing §SAT11 . . . 122
8.4.4 Results of Testing §FUNC3 . . . 123
8.4.5 Results of Testing §FUNC8 . . . 123
8.4.6 Results of Testing §FUNC9 . . . 123
8.4.7 Conclusion of Acceptance Tests . . . 123

9 Conclusion **124**

10 Perspectives **126**
10.1 Adjustments Before Flight . . . 126
10.2 Improvements of the Design . . . 127

Bibliography **129**

A Appendix **135**
A.1 Methods Used in the Project . . . 135
A.1.1 Project Phases . . . 135
A.1.2 Report Structure . . . 138
A.1.3 Tools Developed for Creating the Project . . . 140
A.2 The CAN Bus . . . 141
A.2.1 Data Link Layer . . . 141
A.2.2 Physical Layer . . . 142
A.3 Clock . . . 145
A.3.1 Introduction to PLL . . . 145
A.3.2 PLL - External RC Network . . . 145
A.3.3 Capacitors Connected to the Oscillator Pads . . . 146
A.4 Determination of Resistor Values . . . 147
A.4.1 Resistor Values for Chip Select Pins . . . 147
A.4.2 Resistors Values for Remaining Connections . . . 149
A.5 PCB Design Background . . . 150
A.5.1 Ground Bus Noise . . . 150
A.5.2 Power Bus Noise . . . 150
A.5.3 Transmission Line Reflections . . . 151
A.5.4 Crosstalk . . . 152
A.6 eCos - The Chosen Operative System . . . 154
A.6.1 Semaphores . . . 157
A.6.2 MUTEX . . . 158
A.6.3 Device Drivers . . . 158
A.7 Wait States . . . 159
A.8 Waveforms on Timing Diagrams . . . 161
A.9 Default Settings on the Image Sensor . . . 162
A.10 User Manual for AAUSAT-III CAM . . . 163
A.10.1 Connections to the Camera System . . . 163
A.10.2 Jumper Settings . . . 168

Index **169**

B Flowcharts **173**
B.1 Description of the UML Active State Chart Figures Used in Flowcharts 173
B.2 Boot Sequence 175
B.3 MAIN_thread 177
B.4 ComC_thread 179
B.5 int cam_setup(void *data, U8 length); 181
B.6 int cam_defaults(); 183
B.7 int cam_capture(int time, bol thumb, bol raw) 185
B.8 int cam_list_images(); 187
B.9 int cam_delete_image(int img_id); 189
B.10 int cam_send_img(int img_id, int chunknum, U8 type) 191
B.11 int cam_housekeeping(U16 *hk) 193

C Schematics **195**

D Attached CD **199**
D.1 Contents of CD 199

1 Introduction

In February 1999 the first Danish satellite, The Ørsted Satellite, was launched. It was the culmination of six years of development done by many different participants including Aalborg University (AAU), whom delivered the Attitude Control System (ACS) for the satellite [Bhanderi, 2006]. It provided some of the first Danish satellite technology, and the AAU space program was launched.

The AAU space program involves projects developed by students. The development is done through the students educational projects and upon a freely basis in their spare time. Apart from the primary objective of educating students, it also draws to it a lot of attention, which gives AAU exposure in the media, and is as so regarded as an asset for AAU.

The current space program includes development of pico satellites. These satellites are developed by students at AAU. So far a satellite named AAU CubeSat has been launched in 2003 [Kramer, 2006], and another, AAUSAT-II, is in the early integration phase, and is to be launched in fourth quarter 2006 [Bhanderi, 2006].

AAU has so far, in joint venture with other universities, implemented the knowledge gained from the AAU space program to develop the European SSETI Express satellite. This satellite proved that a lot of the preparations for AAUSAT-II was paying off, as it was the systems developed for AAUSAT-II that was adapted to SSETI Express.

On AAU CubeSat the payload was a camera. However, the mission did not succeed in transferring an image to the Earth before the satellite ran out of power. Another similar camera was sent to space with the SSETI Express, but once again no image was transferred. On the upcoming AAUSAT-II the payload is a gamma radiation sensor and no camera will be implemented.

Now, a third AAU satellite is being developed and is planned for launch in 2008-2009. It is in the very early stages of its mission analysis phase; this satellite is expected to be triple the size of a standard CubeSat, filling an entire PPOD. It has been suggested to equip this satellite with a camera, as a low priority subsystem on the platform. Also a suggestion is made for the payload based on a camera, so it is of great importance to investigate the possibilities of using a camera on the platform.

1.1 Purpose

The camera on the AAU CubeSat was developed and sponsored by an external company, and as a result it will probably be quite expensive to buy a new camera again, since the company would have to re-initiate the production. The purpose of this project is to design a new digital camera for AAUSAT-III, which must be cheaper to implement than the original AAU CubeSat camera and yet it must still be able to function within a pico satellite. Having designed a camera in-house also makes it possible to modify it easily when the mission for AAUSAT-III is finally defined.

The purpose of including a digital camera on the satellite could be to photograph the Earth from the satellites position in Low Earth Orbit (LEO) and transfer the captured images to Earth. Images from the satellite would help document the satellites presence in orbit and draw further attention to the space program on AAU. The current payload suggestion is to photograph the stars, and use the images for analysis of the light emitted. However, as the two areas are somewhat similar in relation to designing the electronics it has been chosen to design a camera for photographing the Earth.

The area on the surface of Earth to be photographed is selected to be up to 100 km wide with a maximum pixel size covering 100 m. The area size is chosen because it will be possible to recognize a part of Denmark. A smaller area would make it possible to get more details, but would require a greater magnification in the lens system and rely more on the precision of position and angel of the satellite.

1.2 General System Description

The system, in its context, is divided into smaller parts and its functionality will be described from the block diagram shown in Figure 1.1.

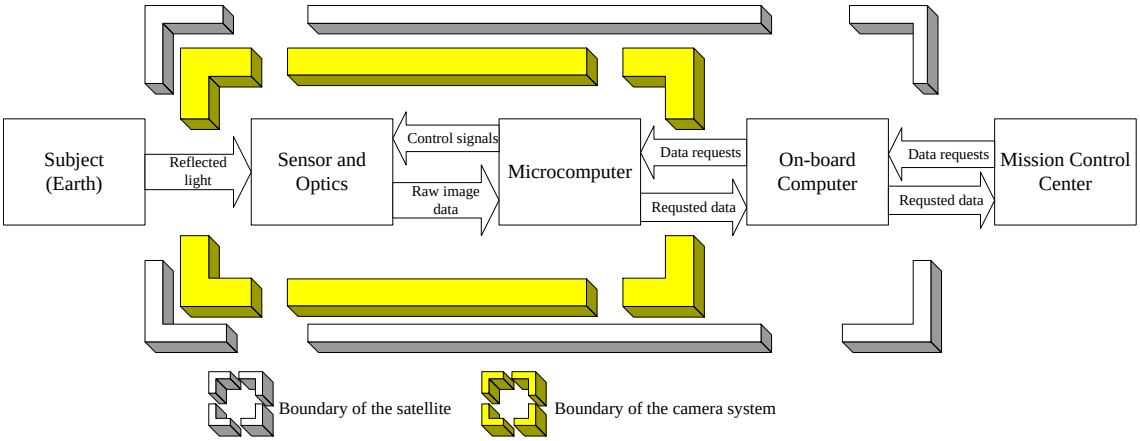


Figure 1.1: *An overview of the parts creating the functionality of the system. Other systems located on the satellite are not drawn into the figure.*

To describe how the system works, the function of each block from left toward the right side of the figure will be described in the following.

- Subject
  Light from the sun shines onto the subject (an area of the surface of the Earth). The surface will absorb some of the light and will reflect the rest.

- Sensor and Optics
  The sensor will convert light reflected by the subject into digital image data. Light is optically focused on the subject and will thereby catch some of its reflected light. Inside the camera, the light is gathered onto an image sensor that converts the light into electric signals. The electric signals are digitalized so they can be processed by the microcomputer.

- Microcomputer
  It is decided due to relative low bandwidth on the Earth to satellite spacelink, to compress the captured images in space. It is also expected that the on-board computer (OBC) will have plenty of tasks to calculate during image processing, and that the OBC is derived from the AAUSAT-II design, not giving it available memory for storing images. Therefore, the image compression is to be done locally at the camera subsystem. The microcomputer that is to be designed shall handle various operations of the camera. It provides the control of the camera. It processes the raw image data from the camera part and provides the storage capabilities for the processed images. It also includes an easy to use interface to the Command and Data Handler (CDH) of the satellite.

- On-board Computer
  The on-board computer is the central computer of the satellite. It runs the CDH software. All communications to Earth is established through this system. Control commands addressed to the camera from Earth will be passed on to the camera system and image data from the camera system is passed on to the Mission Control Center (MCC) on the ground station (GND). The communication channel will not be constant since it depends on the satellites location in orbit.

  The OBC is not a part of the camera system and will throughout the project be treated as an external black box. Since the OBC for the AAUSAT-III has not been designed or build yet, the system will be mainly based on the on-board computer used on AAUSAT-II.

- Ground Computer
  The ground computer has a connection to the satellite and provides a user interface to the camera system. As described in the OBC block, the connection is not constant. The user interface lists the possible interactions with the satellite and passes the corresponding command based on the choice of the user. Furthermore, it must be possible to save the image data on the computer. The final user interface will not be designed in this project. Instead a test platform with a temporary user interface will be established.

  The user should also have the possibility to point the camera by controlling the orientation of the satellite through the user interface. This is, however, not controlled by the camera system itself. Orientation rely on OBC and other subsystems like ADCS that in this project are treated as black boxes. Throughout the project it will be assumed that the satellite is oriented at the desired subject.

Before the required parameters to each block can be determined, it is necessary to understand what parameters exist and how they are adjusted. Therefore, it will be described in detail, how the two blocks (from Figure 1.1) in the camera system work and what the parameters means. The description is divided into six parts.

- Radiation in Space
  Describing the environment that the electronics are to function in.
- Camera Structure
  Describing the process from subject to image sensor.
- Image Sensors
  Describing how light absorbed on image sensors are converted into analog electric signals.
- Microcomputer Hardware
  Describing the possible architectures to base a microcomputer on.
- Software Description
  Describing the intended functionality of the software by means of use cases.
- Communications Protocol
  Describing the protocol used on AAUSAT-II to implement a very reliable protocol for communication with CDH.

2 Analysis

2.1 Radiation in Space

Electronic hardware in space must be able to withstand a larger amount of radiation than electronic hardware on Earth. If the hardware cannot resist the radiation, different incidents can occur. The effects of radiation and possible hardware and software protections against radiation are described in this section.

AAUSAT-III is going to operate in LEO, which means that the environmental radiation level is greater than on Earth, because the Earth is protected against radiation by its electromagnetic field. The electromagnetic field of the Earth has made two major radiation belts called the Van Allen belts, which consist of trapped electrons and protons. The lowest, and smallest Van Allen belt has peak intensity at an altitude of 2000 to 3000 km. This radiation belt has an altitude of approximately 1000 km near the poles, which creates a higher radiation level here. The radiation belt is asymmetric, and therefore has its lowest altitude above the South Atlantic. This creates a region with intense proton flux, called the South Atlantic Anomaly. The outer Van Allen belt has peak intensity at an altitude of 10,000 to 20,000 km [Forslund et al., 1999, p. 36]. The consequences of this radiation are explained in the next section.

2.1.1 Consequences of Radiation

When electronic hardware is exposed to radiation three types of radiation effects can occur; they are explained in this section.

Single Event Effects

Single Event Effects, SEE, can be divided into three different effects: Single event upset, single event latchup, and single event burnout. The three effects are now described [Holbert, 2006].

Single Event Upset, SEU, is a radiation-induced effect that causes errors in electronics, because the radiation can create electron-hole pairs in the electronic circuits. SEU is a nondestructive error, which means that the device can return to normal behavior after a reset. SEUs typically result in bit flips in memory cells or registers. A bit flip is when a bit flips form binary 1 to 0 or vise-versa, corrupting data.

Single Event Latchup, SEL, can create loss of device functionality due to a heavy doze of ions or protons. SELs causes a larger operating current than specified to flow, which can permanently damage the device or the power supply. If this does not happen, the device can return to normal behavior after a power off.

Single Event Burnout, SEB, is a destructive error caused by a heavy doze of ions, which causes the device to fail permanently. SEBs can cause power transistors to conduct a high current, bits to freeze, and create noise in CCD chips. High current in a power transistor can cause device destruction.

Total Ionizing Dose

Total Ionizing Dose, TID, is a radiation effect that first increases the standby current and then decreases retention time of DRAMs. This happens because the threshold voltage on MOS transistors changes when exposed, and leakage currents at the edge of NMOS transistors are generated. The reason why this happens is that charges build up in the gate insulator [Poivey et al., 2002].

Displacement Damage

Displacement Damage is a radiation effect that displaces atoms. The displacement of atoms in for example a silicon lattice causes malfunction. Bipolar devices can be very sensitive to this effect, whereas CMOS devices are not sensitive to this effect [Christiansen, 2004].

2.1.2 Hardware Protection Against Radiation

In LEO the amount of radiation is between 19 krad and 190 krad as shown in Figure 2.1. The amount of radiation dictates how well the electronics should be protected.

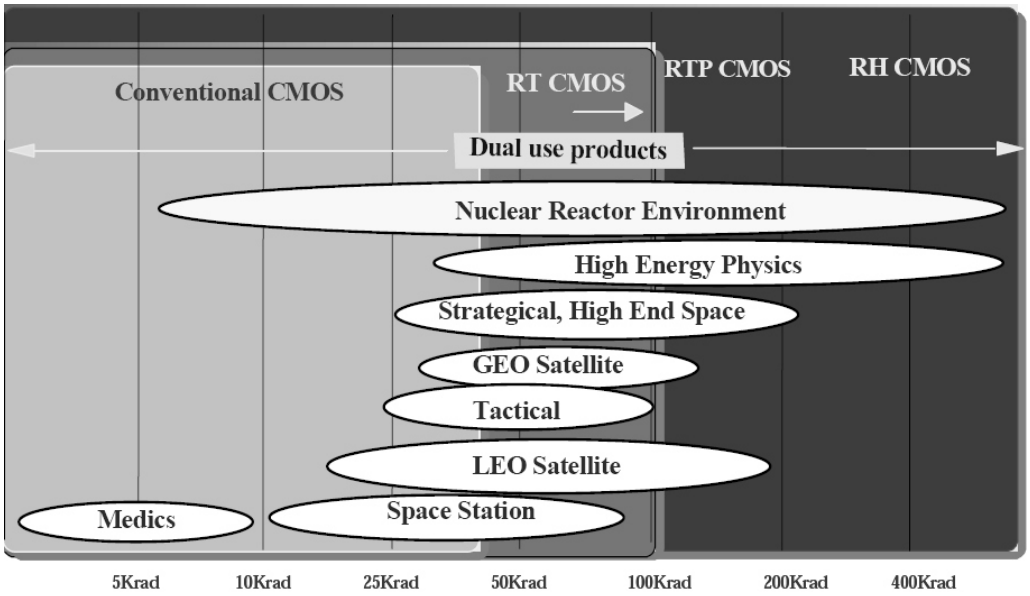


Figure 2.1: *Amount of radiation in different situations [Atmel, 2003, p. 4].*

There are different ways of protecting the electronics and their functionalities from radiation effects. The first is to use radiation-hardened electronics and the second way of protecting the electronics is to shield the electronics from radiation. Both types of protection is not necessary in LEO, because the amount of radiation in LEO in not very high. Radiation-hardened electronics is designed to withstand radiation, but is more expensive that commercial electronics, because special fabrication methods are needed to achieve this property.

To shield the electronics from radiation, metal plating is used. The problem with this approach is to make an adequate shielding, while maintaining a low mass. A CubeSat is launched from a PPOD, and restrictions on mass must be upheld.

Unhardened commercial electronics can typically survive 3-10 krad of total dose without much degradation. Unhardened commercial electronics can typically remain functional from 10 krad to 30 krad [Benedetto, 1998, p. 2]. Although the electronics remain functional it may suffer from high amounts of SEE.

The amount of radiation changes rapidly with the altitude. This means that at altitudes above 900 km shielded commercial parts may become unusable because of the increased radiation. The increased radiation demands heavy shielding [Benedetto, 1998, p. 4].

Most of the camera system is shielded against radiation by the external structure of the CubeSat. According to Figure 2.2 the shielding of a satellite with an altitude of 600 km needs about 0.5 mm aluminum to reduce the radiation level to about 3-10 krad. Therefore, commercial parts can typically withstand the radiation inside the pico satellite as long as the thickness of the MECH is greater than 0.5 mm.

To avoid radiation-hardening the image sensor, a lens system that hides the image sensor from direct exposure to space environment must be designed.

If the casing of the components are made from plastic the PCB has to be covered in Solithane 113300, because Solithane 113300 reduces the total mass loss [Chang, 2002, p. 9].

To make sure that bit flips do not damage the functionality of the satellite it is recommended to make some software error-correction, which is explained in the next section.

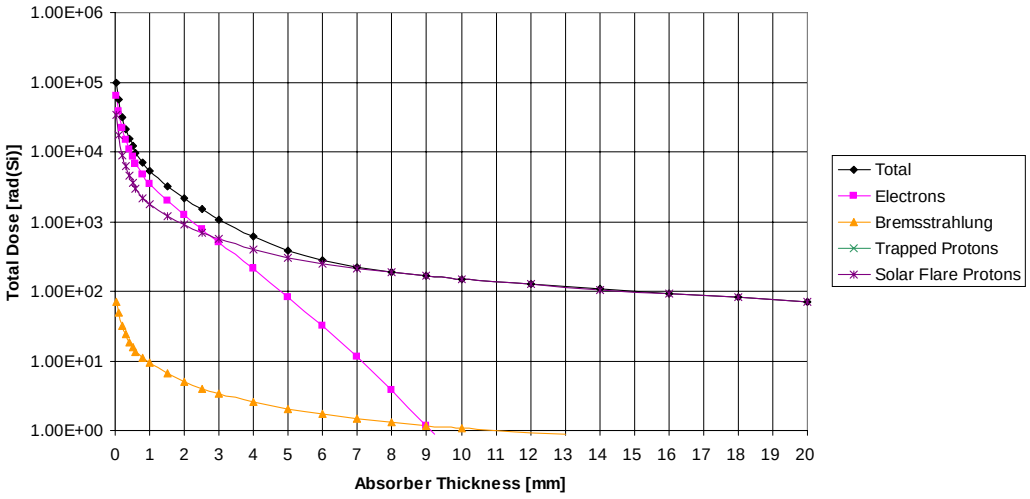


Figure 2.2: *Shielding against radiation at an altitude of 600 km [Hansen, 2001, p. 6].*

2.1.3 Software Protection Against Radiation Consequences

By using an error-correction algorithm to encode data it is possible to recover erroneous data, caused by e.g. radiation. By implementing such algorithm into software running on the camera system, the risk of data being corrupted can be reduced. Error-correction encoding is relevant for the sake of recovering data affected by bit-flips in local data storage.

The ability to recreate data depends on the error and the power of the error-correction algorithm. The Hamming coding is an example of such algorithm, and it will be described in the following text.

Hamming Coding

Hamming Code is an error-correcting code that will correct any single-bit error and will detect any double-bit error [Wagner, 2002]. In a Hamming Code a word of length $m + r$ bits is formed by the sum of an m-bit word and r parity bits. A parity bit is appended to a group of bits to make the sum of all the bits always odd or always even. In a Hamming Code the leftmost bit is numbered 1. Original data bits are moved, thus parity bits are found on bit numbers, which are powers of 2.

To illustrate the Hamming Code an example will be given starting from a 16-bit word [Tanenbaum, 2006, p. 76]. Thus 5 parity bits are added and the original bits containing data are moved sideways such as positions 1, 2, 4, 8 and 16 are occupied by the parity bits. In this example even parity will be used; i.e. the total number of '1's in control lists are even. A bit of position N is controlled by parity bits which have positions i, j and $\cdots$, where $N = i + j + \cdots$. For example b_{14} is checked by parity bits on positions 2, 4 and 8 because $14 = 2 + 4 + 8$. The bit positions checked by parity bits are shown below from the definition made by Hamming [Tanenbaum, 2006, p. 76]. How the bit positions are used to find an erroneous bit are describes later in this section.

Parity bit (p)	p	p		p				p								p					
Ctrl 1	1		3		5		7		9		11		13		15		17		19		21
Ctrl 2		2	3			6	7			10	11			14	15			18	19		
Ctrl 4				4	5	6	7					12	13	14	15					20	21
Ctrl 8								8	9	10	11	12	13	14	15						
Ctrl 16																16	17	18	19	20	21

Ctrl i is a parity check carried out for the bits listed to the right of Ctrl i. Bit b_{i} enters into the binary representation of the position of the bits examined.

In this example it is illustrated how a Hamming Code is constructed for the 16-bit word 1111 0000 1010 1110. Below the Hamming coded word is shown together with an instance containing an error in shape of a flipped b_{05}.

Bit number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Hamming coded word	0	0	1	0	1	1	1	0	0	0	0	0	1	0	1	1	0	1	1	1	0
Word containing error	0	0	1	0	0	1	1	0	0	0	0	0	1	0	1	1	0	1	1	1	0
					↑																

The total number of '1's in both parity and data bits checked by Ctrl 1, 2, 4, 8 and 16 should be even because even parity is used; but the numbers are uneven for the bits checked by Ctrl 1 and 4, which appears in the following overview.

Bit number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	'1's
Ctrl 1	0		1		0		1		0		0		1		1		0		1		0	uneven
Ctrl 2		0	1			1	1			0	0			0	1			1	1			even
Ctrl 4				0	0	1	1					0	1	0	1					1	0	uneven
Ctrl 8								0	0	0	0	0	1	0	1							even
Ctrl 16																1	0	1	1	1	0	even

The bits b_{01}, b_{03}, b_{05}, b_{07}, b_{09}, b_{11}, b_{13}, b_{15}, b_{17}, b_{19} and b_{21} are checked by Ctrl 1, i.e. parity bit 1, and the fact that the total numbers of '1's is uneven means that one of those bit must be incorrect. The total number of '1's checked by Ctrl 4, i.e. parity bit 4, is incorrect too, meaning that one of the bits b_4, b_5, b_7, b_{12}, b_{13}, b_{14}, b_{15}, b_{20} and b_{21} are incorrect. The erroneous bit must be one of the bits in both control lists, i.e. bits b_5, b_7, b_{13}, b_{15} or b_{21} as indicated below.

Bit number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Ctrl 1	0		1		0		1		0		0		1		1		0		1		0
Ctrl 4				0	0	1	1					0	1	0	1					1	0
Repeated bits					0		1						1		1						0

The erroneous bit can be detected and corrected in case of single error event. Thus the erroneous bit can be detected by adding the numbers for the incorrect control lists, i.e. adding the bit numbers for the parity bits concerned. In this case these bit numbers are 1 and 4 result in b_{05} being the erroneous bit. This is possible because of the following description. Bits b_{07} and b_{15} can be eliminated because Ctrl 2 is correct. Similarly, Ctrl 8 and 16 are correct, eliminating b_{13} and b_{21}. Only b_{05} is left means that this bit is incorrect. Since b_{05} was read as a '1' the correct value is '0'.

In practice Hamming Code is handled in way of matrix notation. A Hamming Code can be constructed from a generator matrix consisting of an identity matrix and a parity generation matrix. The original data can be validated by applying a parity check matrix to this result [USNA, 2001].

Code-correction is one way of handling bit errors. Another way is to limit the damage caused by errors in data. In connection with the camera system it is decided to split up image data into a number of chunks before transmitted to GND. This makes the transmitting more flexible, because image data is divided into smaller data parts, which separately or several at a time can be transmitted to GND. Besides, the consequence of single chunk errors will be limited, because GND can request re-transmission of corrupted chunks without being forced to re-transmit an entire image. Corrupted chunks can be unveiled by a corrupted checksum for the chunks concerned.

It has been investigated that radiation can cause different types of radiation effects, which can cause either unharmful effects or in some cases device destruction. Radiation in LEO is above the normal rating for electronics, but for equipment protected by the MECH, it should be possible to use commercial components. For components that could be exposed directly to radiation, radiation hardened electronics should be chosen. Protecting the hardware can be done by shielding components or using radiation hardened components able to withstand a larger amount of radiation. Software protection can be realized by Hamming coding data using parity bits, and introduce error-correcting code being able of correcting any single-bit error. In the following section the structure of a camera will be explained, hence the knowledge afterward can be combined into a camera system in space.

2.2 Camera Structure

This section explains the necessary structure to lead reflected light from the desired subject onto the image sensor. The parts necessary for a digital camera to capture an image are shown on Figure 2.3 and these will be explained. The position of the camera and subject will be described. Afterward,

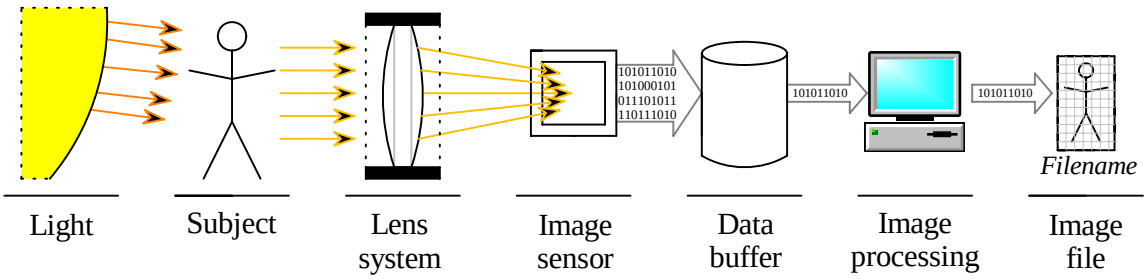


Figure 2.3: *Overview of the parts used in the image capturing process of a digital camera. It can be seen how light fall on the subject and get reflected by the subject. The light reflected are through a lens system focused onto the surface of the image sensor. The image sensor converts the light into raw image data, which is put through a data buffer to a computer. The computer handles image processing to generate the final image file.*

the purpose, principle and calculations of a lens system are explained and photography expressions in relation to the lens system are introduced. Structures of image sensors and microcomputers will be explained in respectively Section 2.3, page 28, and Section 2.4, page 32.

2.2.1 Light Intensity of the Subject

As mentioned in the introduction in Section 1.1, page 16, the camera system will be designed to let the satellite take images of the Earth from Low Earth Orbit at an altitude of approximately 700 km above surface [AAU, 2002, p. 3]. To obtain usable images it is important to ensure the right exposure. This requires knowledge of the light intensity, the camera will experience from the subject. The light seen by the camera is light reflected from Earth. The typical value for direct sunlight is 100,000 lux to 130,000 lux [Micron, 2006]. The light reflected from Earth is approximately 30 % [ASE, 2000, p. 11]. Light intensity does not differ much from the intensity on the surface of

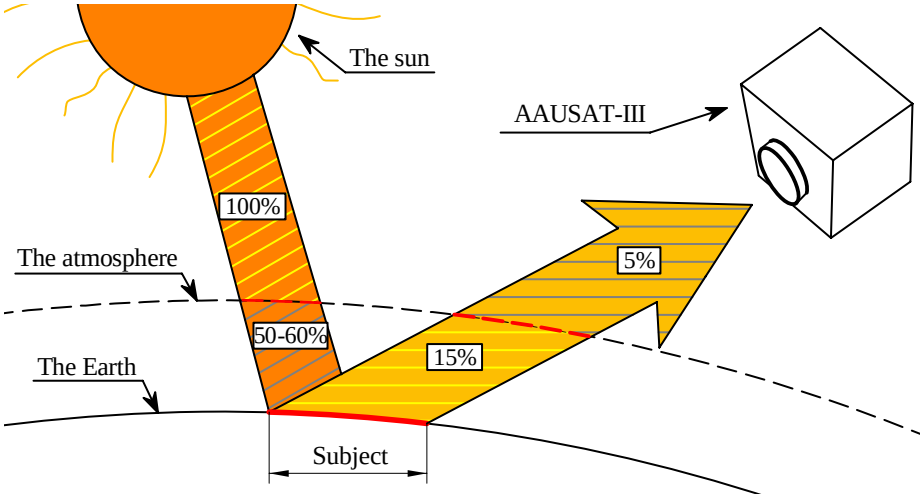


Figure 2.4: *Light reflected from the subject to the camera of AAUSAT-III inspired by [ASE, 2000, p. 11].*

the Earth, it is not expected necessary to take any extraordinary considerations to the amount of light.

2.2.2 The Lens System

The purpose of including a lens is to make sure that only the light beams reflected by the desired subject is projected onto the image sensor. The lens should match the desired subject to the image sensor by gathering the light beams reflected by the desired subject and focus these light beams onto the light sensitive area of the image sensor. The lens system is necessary to meet the aim of capturing an image of Earth with a width of 100 km.

Principle of a Lens

In Figure 2.5 it is shown how a lens redirects parallel light beams. The light that is reflected by the subject onto a lens system equals an input signal and the image created by the lens system equals an output signal. It should be noted that incoming light beams would have different angles unless the distance to the subject is infinite. For the camera system of this project this approximation is fair.

When the light beams passes through the surface of the lens they travel from one medium into another and this causes refractions to occur. Refractions are the bending of waves due to the changing speed of light. The amount of refraction depends on the refraction-index of the media along with the angles between the light beams and the surfaces of the lens. The curved lens shown in Figure 2.5 is thicker in the middle and thinner at the edges; this defines a convex lens. A convex lens makes the light beams bend inward [Serway and Jewett, 2004, p. 36].

On Figure 2.5 it is shown how the light beams cross each other in a single point before they are projected onto a surface where they create the final image. The point where the light beams cross is called the focal point and the distance from this point to the center of the lens is called the focal length. This is not the definition of the focal length; however, it is valid under assumptions of a thin lens where the thickness of the lens can be neglected. Because the light beams cross each other at the focal point a resulting two-dimensional image will be rotated 180°, since a flip happens for each dimension.

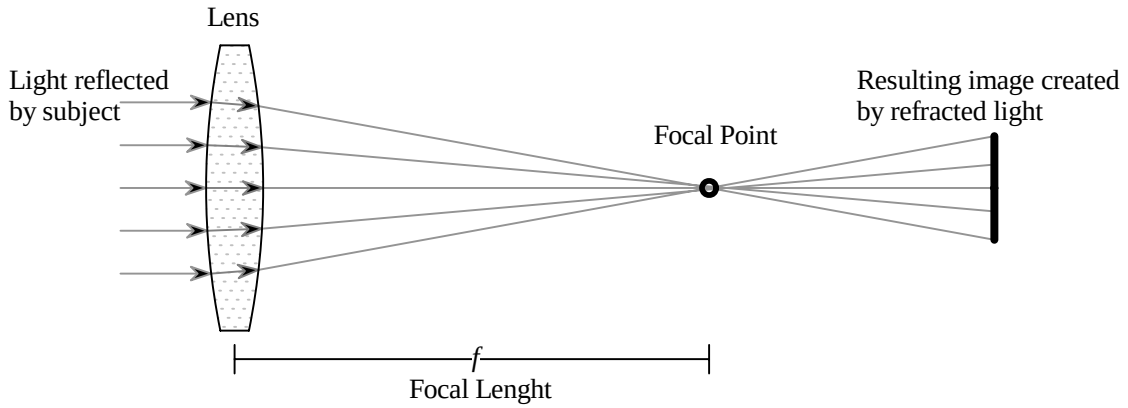


Figure 2.5: *Here it is shown how the light is refracted by the lens and creates a deformed image.*

Calculating the Magnification of a Lens

To explain calculations of a lens system, the ray diagram of a thin lens on Figure 2.6 is used. The image distance q is projected from a point at a distance p of the lens. The projection is based on the focal length f. The example in Figure 2.6 is based on simplified calculations, which are usable as long as the lens is thin and the thickness can be ignored [Serway and Jewett, 2004, p. 1153]. The first ray of the ray diagram passes perpendicular to the front of the lens, is refracted in the lens and passes through the focal point on the image side. The second ray is drawn through the center. Because it passes the center the ray will leave the lens with the same angle and because the thickness is ignored it pass directly through. The third ray passes through the focal point of the subject side and is refracted. The rays intersect where the resulting image is in focus. Because the

three rays start at the top of the subject and the image is flipped, they intersect at the bottom of the image [Serway and Jewett, 2004, p. 1145].

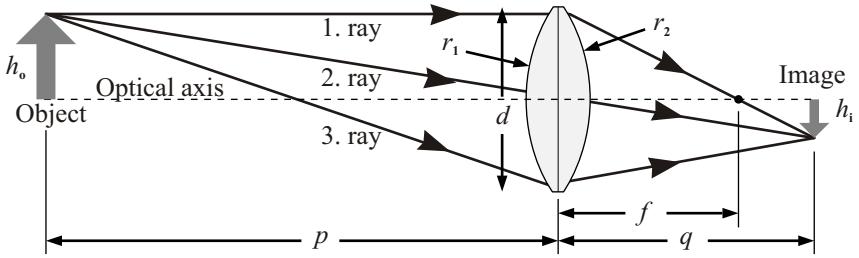


Figure 2.6: *Image formation of a lens. With a ray diagram it is shown how light reflected from a point at the subject is refracted by the lens to form a point at the image. The point where the rays intersect indicates the image position.*

The distance from the lens to the image, q, can be calculated by means of the magnification equation shown in Equation (2.1) [Ryer., 1997, p. 18].

$$m = \frac{h_\mathrm{i}}{h_\mathrm{o}} = -\frac{q}{p} \qquad [\,\cdot\,] \tag{2.1}$$

where:
m is the magnification $[\,\cdot\,]$.
h_i is the height of the image [m].
h_o is the height of the object [m].
q is the distance from the lens to the image [m].
p is the distance from the object to the lens [m].

The negative sign represents a flipped image.

The focal length, f, can be calculated by means of the Lens equation showed in Equation (2.2) [Ryer., 1997, p. 18].

$$\frac{1}{f} = \frac{1}{q} + \frac{1}{p} \qquad \left[\frac{1}{\mathrm{m}}\right] \tag{2.2}$$

where:
f is the focal length of the lens [m].

The focal length can be determined by three parameter according to the Lens Maker's equation, Equation (2.3), [Ryer., 1997, p. 18]. A lens with a large curvature will converge the light beams at a point close to the lens. For a lens with a flatter shape light beams will not turn as sharply.

$$\frac{1}{f} = (n-1)\left(\frac{1}{r_1} - \frac{1}{r_2}\right) \qquad \left[\frac{1}{\mathrm{m}}\right] \tag{2.3}$$

where:
n is the refraction index of the material $[\,\cdot\,]$.
r_1 is the outside radius of the lens [m].
r_2 is the inside radius of the lens (negative sign convention for radius) [m].

Combining Lenses

By putting more lenses on a row the magnifying effect is increased. The image created by the first lens will serve as a virtual subject of the second lens. The magnification equals the product of the magnification of each lens. The resulting theoretical focal point is shown in Equation (2.4) [Serway and Jewett, 2004, p. 1149].

$$\frac{1}{f} = \frac{1}{f_1} + \frac{1}{f_2} \qquad \left[\frac{1}{\mathrm{m}}\right] \tag{2.4}$$

where:
f is the resulting theoretical focal length [m].
f_1 is the focal length of the first lens [m].
f_2 is the focal length of the second lens [m].

Calculating Aperture of a Lens

When a lens system is used for a camera, the lens or lenses are mounted in a light-sealed case in front of the image sensor. The lens system makes it possible to ensure that the image sensor is only exposed to light reflected from the desired subject. By controlling the aperture of a lens system it is also possible to control the amount of light. The aperture of a lens system is a diameter which limits the incoming light beams. The diameter of a lens is an example of an aperture. Another aperture in the same lens system could be the cabinet holding the lens, if it block the light beams. The limiting aperture is called the aperture stop. The effective aperture refers to the effective size of the light opening of the lens. When it is measured in relation to the focal length, it is called f-number, see Equation (2.5) and Figure 2.7 where it is illustrated [Wikipedia, 2006d].

$$f/\# = \frac{f}{D} \qquad [\,\cdot\,] \tag{2.5}$$

where:
$f/\#$ is the f-number of lens $[\,\cdot\,]$.
f is the focal length [m].
D is the diameter of the lights opening called effective aperture [m].

The f-number symbol is by convention written as $f/\#$ and must be treated as a single symbol. When the value is specified it is written instead of the # [Wikipedia, 2006d]. Thereby, a $f/\#$ is actually a constant with its value written in the notation. As example $f/2$ is used instead of $f/\# = 0.5$. When the diameter of the aperture, D, in Equation (2.5) increases, the # relative to the focal length decreases.

The aperture of a lens system can be limited by placing a disc with a small hole between the image side of lens and the image sensor. The radius of this hole will be smaller than the radius of light led in by the lens and thereby limit the amount of light led into through the lens system. This is shown in Figure 2.7. Notice that the lens concentrates the light beams and the effective aperture diameter is larger than the diameter of internal aperture, which limiting the concentrated light. A variable aperture will be able to adjust the radius of its hole. The maximal aperture (and thereby the lowest #) of a lens system is often referred to as the speed of the lens, because it determines how fast-moving subjects the lens will be able to capture without getting a skewed image [Serway and Jewett, 2004, p. 1154].

2.2.3 Exposure of a Camera

Exposure is the time integral of illuminance. When an image is captured by a camera, correct exposure must be ensured to achieve proper illuminated images. If too much light is absorbed by the image sensor, the image will be over-exposed since the dynamic range will be exceeded, and the image will be too bright. This results in a dim image and details cannot be seen.

The following text explains which factors affect the exposure and their relations. The amount of light absorbed on the final image depends on the amount of light reflected by the subject, the exposure time, the aperture of the lens system, and how sensitive the image sensor is. The exposure at the focal plane, where the image sensor is located, can be calculated by Equation (2.6) under assumptions which are valid when the lens is focused at infinity [Standard, 1998, p. 12].

$$H_\mathrm{a} = \frac{65 \cdot L_\mathrm{a} \cdot t}{100 \cdot A^2} \qquad [\mathrm{lux} \cdot \mathrm{s}] \tag{2.6}$$

where:

H_a the time integral of intercepted light [lux · s].
L_a the amount of light radiated from the subject [$\frac{cd}{m^2}$].
t is the exposure time [s].
A is the f-number of the lens system [·].

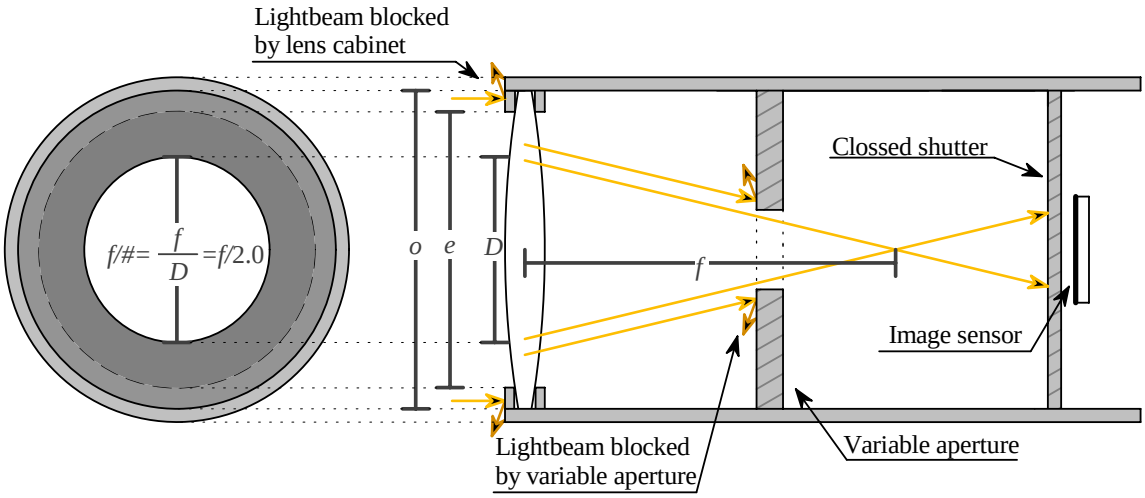


Figure 2.7: *The physical construction of an optical system for a camera is shown. The variable aperture brings the diameter limiting the incoming light. The length o is the diameter of the lens, e is inner diameter of the lens cabinet, and D is the diameter of the circular lens area being used. The calculated $f/\# = f/2.0$ is based on the measures of the figure. Inspired by [Serway and Jewett, 2004, p. 1153].*

The amount of light, which an image captured by an image sensor is exposed to, can be determined by calculating the exposure. From Equation (2.6) it can be seen how the amount of exposure increases linear with both time and the amount of light and decrease quadratic with the f-number of the lens system given that A is the f-number of the lens system.

Controlling the Exposure Time

In order to obtain a sharp image the light reflected from the subject should remain constant on the image sensor during the entire exposure time. If the subject movement during the exposure time is too large, the resulting image will become blurred and it might look like the subject is followed by a tail of light.

The exposure time for a camera can be controlled by placing a mechanical shutter inside the lens system, however, it is dispensable for a digital camera since the exposure of an image sensor can reset, in contrast to a film spool. When using this reset functionality, it is still necessary to control the exposure time, and if a mechanical shutter is unavailable the pixels must be extracted very fast to get an uniform exposure on all pixels. Details of how reset operations can replace a mechanical shutter are explained in Section 2.3.4, page 30, after the principle of an image sensor has been explained.

Sensitivity

The sensitivity of a camera determines how bright pixels will be at a certain exposure. The photoelectric sensitivity of an image sensor is typically rated in $\frac{V}{lux \cdot s}$. However, the intensity of a pixel also depends on the offset level and the gain applied to the image after conversion from exposure to voltage. The amount of these cannot be individually adjusted on a digital camera, but instead the sensitivity as a whole can be set to match a sensitivity rated as an ISO speed. Typical ISO

speeds which can be selected are 100-400. Methods to determined such values are specified by International Standard, equal the sensitivity of a film spool [Standard, 1998]. Digital cameras will produce a larger amount of noise in the image if a higher ISO speed are used and higher gain is applied to the output for the image sensor. Image sensors of high quality or larger pixel size tend to produce less noise, thus allow usable images at higher sensitivities.

Other Relations

It appears from Equation (2.6) how equal exposures can be archived with different settings of aperture and exposure time. However, it should be noticed that the depth of focus depends on which settings are selected. The depth of focus determines how close and how far objects can be located from the subject in focus and still remain sharp. Setting a higher f-number the amount of incoming light will decrease, require a longer exposure time and result in a larger depth of focus. For this project the effect of focus depth are neglected since the distance to the subject are very far and the focus are approximate a focus at infinity.

There is a strong connection between the choice of image sensor and choice of optics, which means that these parts have to be designed from a total consideration. The sensitive area of the image sensor and the size of the subject define the property for the optics. In addition, the lens has an affect on the amount of light exposed on the image sensor etc. Prior to the lens design, the sensitive area of the image sensor must be defined. In order to get the optimum usable data from the image sensor, the entire sensitive area must be exposed.

It is chosen not to implement an optical zoom or a variable aperture. This would involve unnecessary complexity to the structure and involve movable parts. The construction would be larger and require an actuator e.g. an electro motor. Adjustable focus will not be necessary, because the camera will remain at approximately the same altitude during its orbit of Earth. Beside this, the far distance makes the setup of the focus of the camera approximately equal to a focus at infinity.

It has been described how optics leads reflected light from the subject onto the image sensor. In the following section, the conversion from light to electrical signals in image sensors will be explained.

2.3 Image Sensors

The purpose of this section is first to provide a general understanding of different image sensor types and then to compare the different types of image sensors. Digital cameras are mainly using two different technologies of image sensors called CCD, Charge-Coupled Device, and CMOS, Complementary-Metal-Oxide-Semiconductor. The two image sensor technologies will be discussed separately and compared.

2.3.1 CCD

The principle of a CCD image sensor is illustrated in Figure 2.8. Here the photons in light are absorbed by silicon photodiodes. Each photodiode makes a Photon-to-Electron Conversion. The amount of charge, i.e. number of electrons, is proportional with the light intensity. This process causes a charge to build up in each pixel in the CCD image sensor. It is desirable to get a voltage to represent the amount of light absorbed by the pixels instead of a charge. Therefore, an electron to voltage converter is integrated in the CCD image sensor. To get the image represented by voltages, every charged pixel has to be converted into a voltage sequentially. The charge is moved pixel by pixel toward the readout register by manipulating the potential in the pixel area. The potential manipulation in the pixel area is created by applying a voltage to the gate [Benamati, 2003, p. 2].

CCD image sensors have been designed to optimize image quality. This design has made CCD chips unsuitable for efficient integration of other electronics onto the chip [Titus, 2001, p. 3]. This means that CCD image sensors require a lot of control electronics as shown in Figure 2.8. On the PCB of the CCD image sensor the bias voltages, timing and clock is generated, as well as the gain of the signal from the image sensor and a A/D-conversion. Furthermore, some sort of line driver can be necessary too.

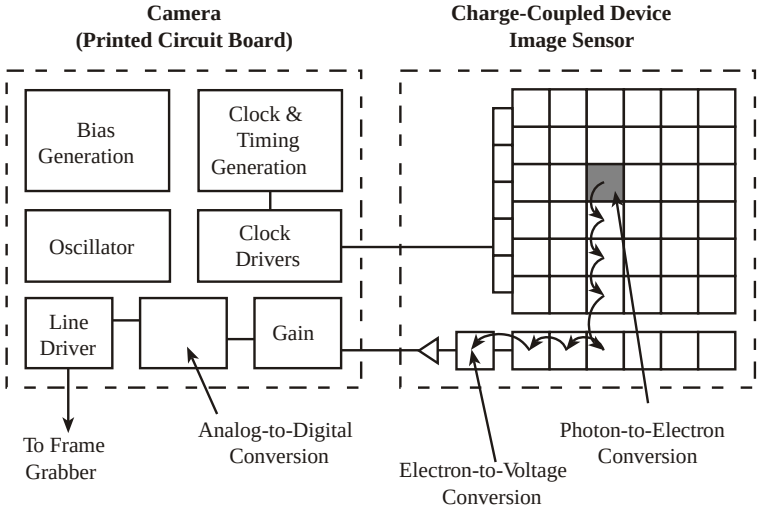


Figure 2.8: *Block diagram of a CCD image sensor [Litwiller, 2001, p. 1].*

2.3.2 CMOS

The principle of a CMOS image sensor is illustrated in Figure 2.9. Photons in light are absorbed by the photodiodes, which convert the light intensity into charges. The charge at every pixel is converted into a voltage by a couple of CMOS transistors connected to each pixel. This design of CMOS image sensors has the advantage that any pixel can be selected and read.

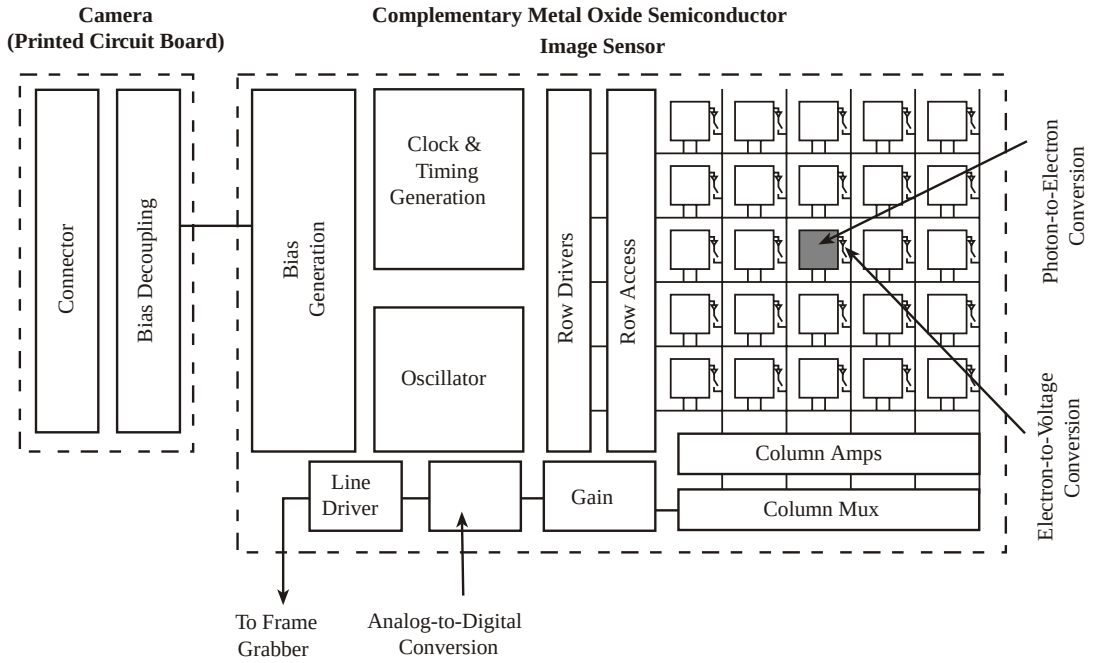


Figure 2.9: *Block diagram of a CMOS image sensor [Litwiller, 2001, p. 2].*

CMOS image sensors are made with standard CMOS silicon processes. Therefore, other functionalities can be included in the CMOS image sensor chip, as shown in Figure 2.9. Here all drivers, clock, timing, biasing and A/D conversion are integrated into the image sensor chip. This means that almost no external components are needed to operate a CMOS image sensor, as shown in Figure 2.9, where only decoupling and a connector are necessary besides the image sensor.

2.3.3 Comparison of CCD and CMOS

The difference between CCD image sensors and CMOS image sensors is generally the amount of on-chip circuitry and the location of the charge to voltage conversion. These differences cause some issues for both image sensor types.

The greater amount of on-chip circuitry in a CMOS image sensor compared to a CCD image sensor results in increased noise on the output signal of the CMOS image sensor. Meanwhile, the greater amount of on-chip circuitry means that a CMOS image sensor only needs one bias voltage, where CCD image sensors on the other hand typically require a few high-voltage biases. The location of the charge to voltage conversion also causes some different issues for the two image sensor types.

A parameter called quantum efficiency is different for the two image sensor types. The quantum efficiency tells how much of the incoming light is absorbed by the sensor, i.e. the ability of the sensor to catch photons and generate electrons from the photons. A CMOS image sensor has at least three CMOS transistors in every pixel, which are optically dead structures. CCD image sensors have no optically dead structures in the sensor, which results in a better quantum efficiency. When comparing Figure 2.8 and Figure 2.9 it is obvious to see that a much smaller area of the CMOS image sensor is covered with photodiodes, than of the CCD image sensor. This means that the light intensity is measured in a smaller area in a CMOS image sensor than in a CCD image sensor.

An issue for the CCD image sensor is charge transfer. The charge in every pixel has to be transferred to the charge to voltage converter. In this process losses can occur, because of damage to the silicon. "*This makes the CCD extremely sensitive to high-energy radiation sources that damage the silicon and induce electron traps-for example, high-energy protons in space.*" [Janesick, 2002].

The last issue described in this section is the charge collection efficiency. The charge collection efficiency indicates how well the charge stays in each pixel. Charge collection efficiency is a larger issue in CMOS image sensors than in CCD image sensors, because the resistance of the silicon in CCD image sensors is larger than in CMOS image sensors. The low resistance of the silicon is required because of the low voltage drive. An overview of the comparison of CCD and CMOS image sensors is shown in Table 2.1.

Pros for CCD and CMOS image sensors	
CCD image sensor	**CMOS image sensor**
Lowest noise	One bias voltage
Highest quantum efficiency	Smallest sensitivity to radiation
Flexible integration	Easy integration of circuitry
Smallest pixel size	Low power consumption

Table 2.1: *Comparison of CCD and CMOS image sensors.*

Both CCD and CMOS image sensors are undergoing an intensive development, where CMOS designers try to achieve higher image quality, and CCD designers try to lower their power requirements and pixel size [DALSA, 2005].

2.3.4 Exposure Reset Operations of Image Sensors

As mentioned in the explanation of exposure time in Section 2.2.3, page 27, image sensors features different reset operations for controlling the exposure time.

Electronic Shutter

Electronic shutters drain the charges in all pixels simultaneously and this global reset makes all pixels start converting light to charges from the same time. Electronic shutters are used for CCD image sensors with progressive scan read-out and interline read-out. CCD image sensors with progressive scan read-out are able to transfer their charge in all pixels simultaneously to light shielded areas of the image sensor and can thereby stop the exposure time without using a mechanical shutter [Kodak, 2003, p. 2]. Interline read-out does not transfer all pixel to light shielded areas simultaneously, but

either all the odd or all the even lines. As so, charges from half of the lines of the image sensor must be extracted before the other half can be extracted. This means that the exposure time will not be equal for odd and even lines if a mechanical shutter is not used, and image sensors featuring only interline read-out cannot capture proper still images without a mechanical shutter. CMOS image sensors supporting global shutter allow a similar operation, where all pixels are simultaneously transferred to light shielded areas of the image sensor before they are extracted [Kodak, 2003, p. 3].

Rolling Shutter

Rolling shutters subsequent reset each line of the active pixel array individually starting from the top and then runs through each line. After a short period the charges of pixels in each line are likewise extracted one line at a time, starting from the top. The reset and read-out of each line progress with an equal speed and the exposure time can be controlled by setting period between the reset and read-out start. This mode of operation is illustrated on Figure 2.10.

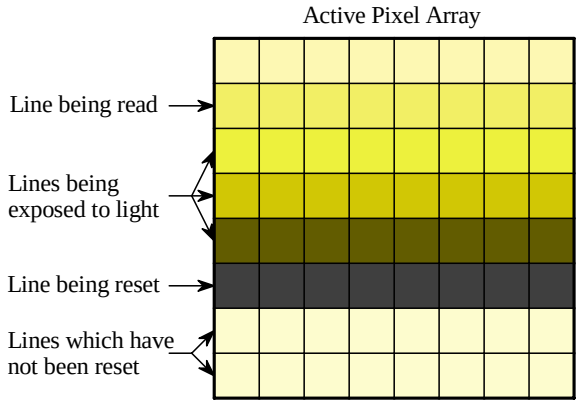


Figure 2.10: *The rolling shutter can be used to control the exposure time of the pixels in the active pixel array by setting the period between reset and read-out of pixel lines. It can be seen how the line of pixels which are reset has not yet intercepted any light and is very dark while the upper lines get brighter and brighter. The example of rolling shutter shown, has an exposure time which equals the time it takes to shift reset and read-out three times. The lines which have not been reset yet are very bright because nothing blocks the light to the image sensor until the moment it is reset.*

The exposure time of each line does not start simultaneously and if it takes too long time to read-out the image, it might create an issue for fast-moving subjects. The problem occurs when a subject has moved in the period of time that occurs from the first line is extracted to the last line is extracted. The subject on the captured image will not have blurry edges or be followed by tails of light like an analogous camera would produce when capturing with too long exposure time, but instead it will be skewed horizontally. To minimize this problem with rolling shutter the read-out must happen as fast as possible, and this requires that a large amount of image data is transferred rapidly. However, it should be kept in mind that higher operation frequencies will make circuits produce more noise and will cause an increased amount of noise in the image. Rolling shutters are used with CMOS image sensors to control the exposure time and make it possible to capture images without using a mechanical shutter.

An image sensor, as described above, is the transducer of the camera system. In order to store and process data a microcomputer system is needed. In the following section functionality and possible architecture of the microcomputer system are described.

2.4 Microcomputer Hardware

In the following section functionality of the microcomputer system and different architecture for a microcomputer system are described.

2.4.1 Hardware Functionality

The electrical signals from the image sensor must be converted into full color image data. It is decided in this project that this job and the task of image processing are to be handled locally in the camera subsystem, so that the OBC is not to waste resources on processing image data. This leads to the following tasks for the hardware:

1. Convert the signals from the image sensor to digital signals that can be handled as data by digital circuits and provide an interface for the image sensor so that the rest of the hardware can control how the image sensor operates.

2. Provide long-term data storage allowing parts of an image, or an entire image, to be retrieved by CDH when required.

3. Establish electrical interface for the camera system, allowing the data to be transmitted between this circuit and the OBC.

4. Provide a structure that will allow embedded software to obtain, process and store image data.

The interfaces of the microcomputer is described before the actual microcomputer is designed.

Microcomputer Interfaces

Image Sensor Controller
The first task, converting analog signals to digital signals and providing an interface to the image sensor, is to be handled by a circuit close to the image sensor, from now on named "Image sensor controller". The image sensor controller should let the rest of the hardware access the image sensor as a device providing data on a data bus. It should also provide a digital control interface to the image sensor allowing the hardware designer to setup the parameters of the image sensor.

Satellite Communication Bus
Internally in the satellite, the OBC is connected to the rest of the subsystems through a shared bus. Like other subsystems, the camera system must communicate with CDH through this bus. There are several bus standards available, and it has not yet been decided which one to use on the AAUSAT-III. However, it is a fair guess that one of the buses already tested on AAU CubeSat and AAUSAT-II will be reimplemented. AAU CubeSat used the I^2C bus standard and the CAN bus is used on AAUSAT-II. Specifications like voltages and timings must comply with the standards used in the rest of the satellite. On AAUSAT-II the CAN is overlayed by the HSN protocol, this is described in Section 2.6, page 38.

Microcomputer

To solve the remaining tasks the microcomputers will have to be designed and an architecture must be chosen. A microcomputer will be constructed to handle image processing, data interface with the CDH and to store image data. There are in general two overall design approaches in constructing microcomputers: The Harvard structure and the Von Neumann structure [Tanenbaum, 1999].

Harvard Structure
The Harvard Structure is basically a design approach where software and data are separated objects. This provides isolated buses for data and software. There are some advantages to be found in this design, since data handling would never be able to corrupt the software. The disadvantage is, however, that the approximate size of the software and the data is to be known before any

components are chosen to provide memory. The system could end up with a lot of empty memory in the software providing chip, and lacking space for data, or the opposite could occur.

An example of the Harvard structure is shown on Figure 2.11. The figure shows how one bus handles data and another bus handles software. This structure is for instance used in DSPs where

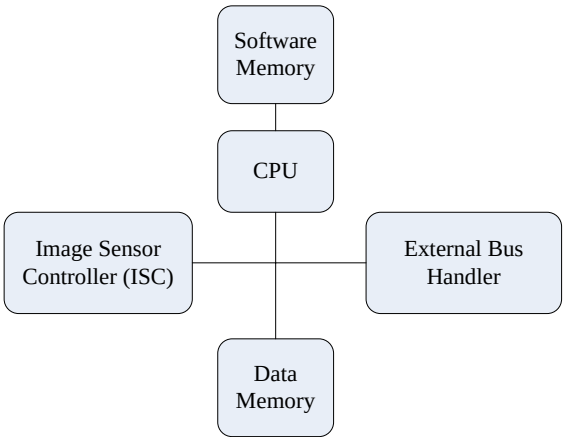


Figure 2.11: *A basic Harvard architecture implemented on the camera system.*

tough real-time demands are found. The reason for using the Harvard structure in such DSPs is that a bus only handling one type of data can be expected to work faster than one handling many types simultaneously.

Von Neumann Structure

The Von Neumann Structure is a design structure where the same bus provides access to both data and software. In relation to the hardware, this will simplify the structure, since the hardware designer will not have to consider what the memory is to be used for. This hardware structure however will require a software design that then distinguishes between software and data. Giving access to both types of memory through software will allow the programmer to define through the software, how much memory is needed. This can be changed easily while running if either software or data needs more memory. In this design, only one data bus is required for all operations. Such

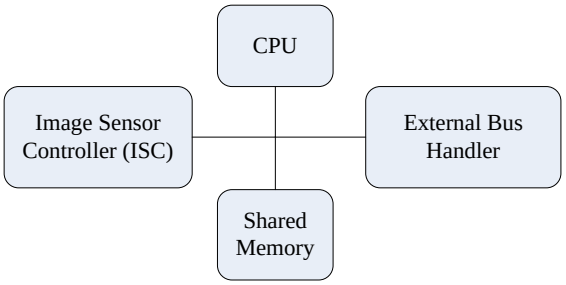


Figure 2.12: *A basic Von Neumann architecture implemented on the camera system.*

a design can be seen on Figure 2.12. This is regarded as the more flexible of the two systems. However, data flow will be expected to be a bit slower than on a Harvard, since the bus here is to handle both software and data.

Address Control Logic

In both architectures addressing of memory and I/O units are usually done by defining an address in the software. As an example, when addressing memory, the address for the memory cells in the memory pool would be transmitted as a binary value on the address bus. The basic CPU does not in any way take care of which specific physical unit stores the memory cell. If the memory is divided over two or more memory chips, the addressing needs to be interpreted by some external logic to

handle which chip is enabled. This logic can be either designed in gates or could be programmed into a logic device such as a PEEL or a FPGA.

Another aspect of address control logic is the constraints of the data bus architecture. As an example, if all data transfers are handled by the CPU, the bus speed is directly dependent of the CPU speed and load. This could in some sessions be a problem. In this project such a timing problem could arise when actually transferring data from the image sensor circuit to the memory cells. Should the image sensor produce data faster than the CPU can properly address it, data could be lost. A way of solving this is to introduce some form of memory control that temporarily takes over control of the bus allowing the external device to gain Direct Memory Access (DMA). This could also be programmed into a logic device or handled by a dedicated DMA controller.

A microcomputer architecture, like described above, is just a platform for executing commands. It is not by itself able to do anything useful without software. If the hardware is designed flexible, it is possible to implement software that can take care of all the communication and data processing. This will be explained in the following section.

2.5 Software Description

The basic model of the hardware structure has been introduced and in this section it will be described, which tasks should be handled by the software running on it. In order to create an overview of the needed software, it must be found out what functionality is needed and it must be split up in smaller tasks.

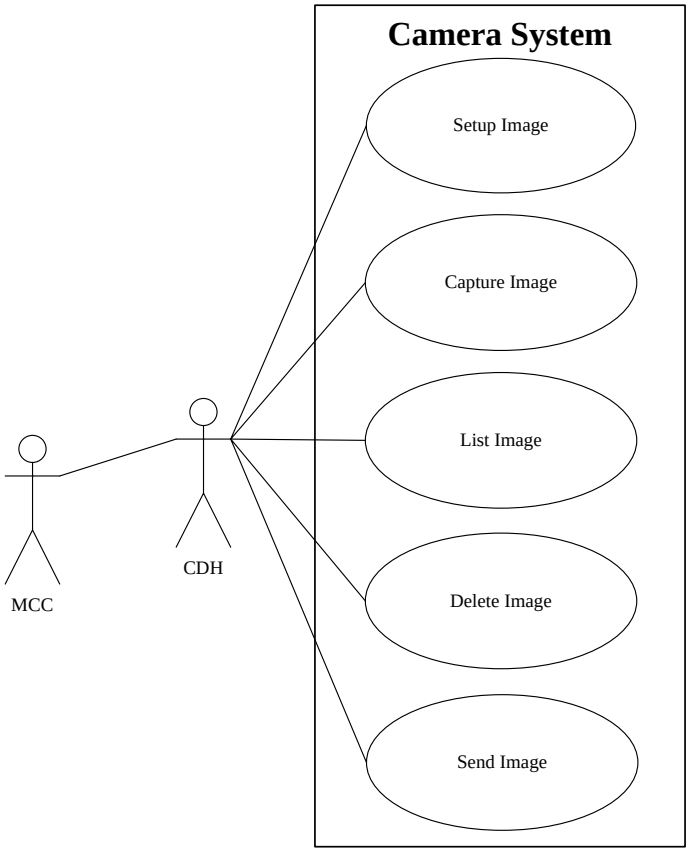


Figure 2.13: *All of the use cases for the camera system shown together.*

2.5.1 Use Cases

The functionality will be specified by means of use cases. Use cases employ functional requirements and are used to build up sequences of actions that a system can perform when interacting with

actors of the system [Larsen, 2006b, p. 34]. The actor could be a user or external hardware. The UML language is used to describe the use cases.

In Figure 2.13 the joint use cases for the camera system are shown.

To understand the following use case descriptions, it is necessary to understand the interaction between MCC, CDH and EPS. Because CDH is the main software unit in the satellite, it is responsible for every major action the satellite is to perform. At launch, CDH will probably not have any prescheduled events concerning the camera, which means that every command for the camera is to be sent from MCC. When GND sends a command from MCC to CDH it can be specified that the command is to be carried out immediately or that it is to be carried out at a given time. If it is not an immediate command, CDH will put the command into the FLP. So when the camera system receives a command it is either directly received from MCC and only relayed by CDH or it has been delayed in the FLP and then later sent by CDH to the camera sytstem.

When the camera system is in the middle of an execution of a command it cannot process another command until the command in progress has finished. If for instance the camera system is in the middle of compressing an image and a command is received, that another image should be deleted, the camera system will proceed compressing the image and will not delete the other image until the compression is done. This implies that the camera system should be capable of buffering a number of commands that have been sent to it.

In the following text, when a description tells that a message is returned, it indicates that if the original command has been sent directly from MCC, the returned message is both logged by CDH and sent to MCC, but if the command was scheduled by FLP the returned message is only logged by CDH. The log can be fetched from MCC at any given time.

Setup Camera

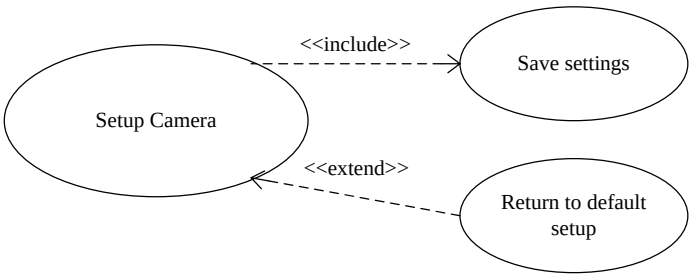


Figure 2.14: *The use case "Setup Camera" with includes and extends.*

- **Purpose:**
  To get new camera settings and save them for later use or return the camera to default settings.
- **Typical scenario:**
  1. CDH either sends a command containing new camera settings or commands the camera to load default settings.
  2. Settings are validated.
  3. The new or default settings are saved as current camera settings.
- **Exceptions:**
  If the settings are invalid they are ignored and an error message is returned.

Capture Image

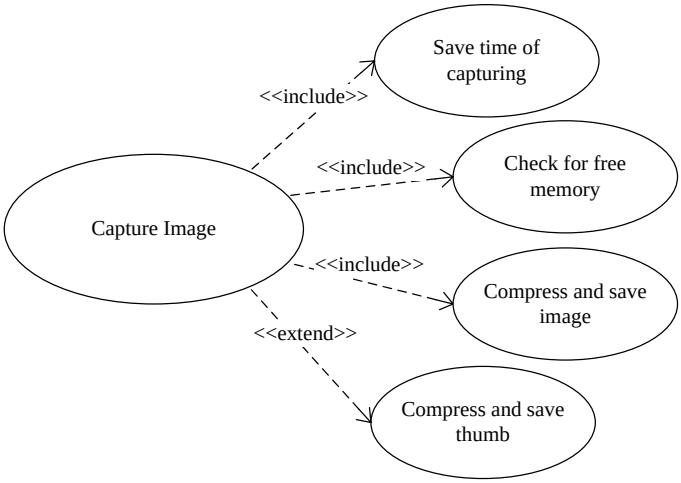


Figure 2.15: *The use case "Capture Image" with includes and extends.*

- **Purpose:**
  To capture an image of what is in front of the camera and to compress and save it to non-volatile memory.
- **Typical scenario:**
  1. CDH commands to capture image.
  2. Check for free non-volatile memory for the final image.
  3. Capture image.
  4. Save time of capturing.
  5. Resize image when thumbnail is to be created.
  6. Interpolate image.
  7. Compress image.
  8. Save image in non-volatile memory.
- **Exceptions:**
  If there is not enough non-volatile memory for the image an error message is returned.

List Images

- **Purpose:**
  To get a list of images in non-volatile memory with time of capture and byte size sent to MCC.
- **Typical scenario:**
  1. CDH requests list of images.
  2. List is generated with image number, time of capture and byte size of each image.
  3. The list is send to MCC relayed by CDH.
- **Exceptions:**
  If the list is empty a message hereabout is returned.

Delete Image

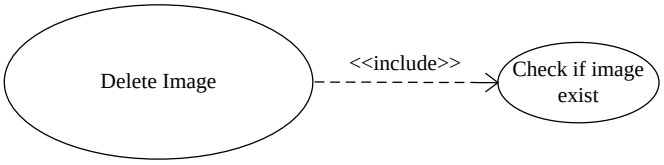


Figure 2.16: *The use case "Delete Image" with includes.*

- **Purpose:**
  To remove an image from the non-volatile memory.
- **Typical scenario:**
  1. CDH commands to delete a specific image.
  2. It is verified that the image exists.
  3. The image is deleted.
- **Exceptions:**
  If the image does not exist an error message is returned.

Send Image

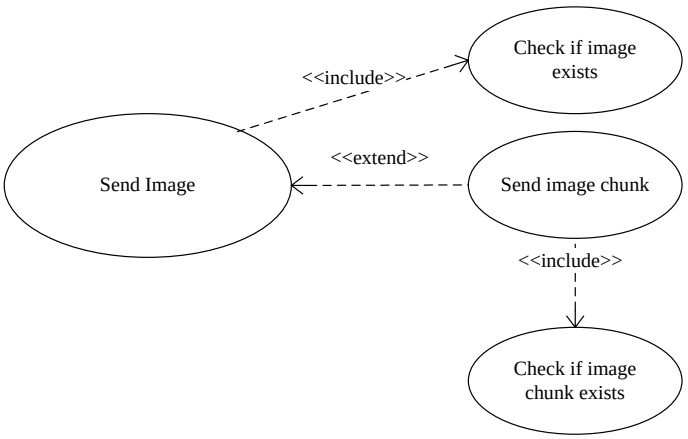


Figure 2.17: *The use case "Send Image" with includes and extends.*

- **Purpose:**
  To send an image or image chunk from non-volatile memory to MCC.
- **Typical scenario:**
  1. CDH requests a specific image or image chunk.
  2. Verify that the image or image chunk exists.
  3. Make data transfer to MCC relayed by CDH.
- **Exceptions:**
  If the image or image chunk does not exist an error message is returned.

In this section the functionality of software has been specified by means of use cases. In the next section HSN is described because AAUSAT-II uses this protocol for internal communication and it is therefore likely that AAUSAT-III is going to use this too.

2.6 Communications Protocol

The Hardcore Satellite Network (HSN) was devised as an alternative to the INSANE protocol developed for the AAUSAT-II projects [03gr731, 2003]. HSN is the protocol that is currently on the subsystems in AAUSAT-II to establish two way communications on the CAN network. The CAN protocol features message transmissions that is checked by the CAN controllers. Multiple error types are detected by the controllers and only packets approved by the CAN controllers are handled by the HSN.

The CAN protocol is further explained in Appendix A.2. Data collisions are not possible on the CAN bus, due to the message priority set in arbitration fields. A CAN packet priority is also used for identification of the packet in the identifier part of the arbitration field.

The HSN utilizes only the CAN identifier fields to setup a communications network on the CAN bus. It provides features that allows for data transmissions of data longer than the 8 bytes that can be put in a CAN packet. This is made possible in the HSN Frame type field in the HSN packet. The usage of this is explained later in this section.

HSN is a simple protocol that implements a communications port number and an addressing on the network. HSN is optimized to run on microcomputers with a very limited amount of memory [Laursen, 2006]. The relation between the CAN protocol and the HSN layer is illustrated on Figure 2.18.

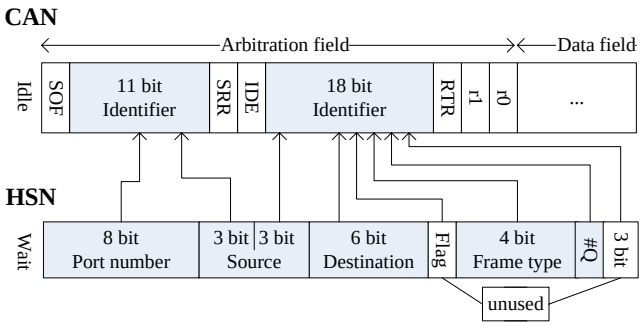


Figure 2.18: *The CAN identifier bits and the corresponding HSN use of them.*

2.6.1 Network Design

The HSN port number is the first part of what is send on the CAN. The individual HSN fields have different tasks to solve, dividing the CAN arbitration fields into several HSN fields allowing for many different information to be given about a HSN packet. A packet as so contains both the protocol information and the actual data that is to be transmitted, the data is unaltered by the protocol.

For a flexible communications design HSN ports are used to define the priority of the commands and data packets on the network. The HSN address defines which subsystem is to reply on the packet. The addresses used define both the Source of a packet and a Destination on the network using the designated address fields, see Figure 2.18.

The memory used by HSN on each subsystem is proportional with the number of ports the individual subsystem has to be active on. Only one HSN packet is buffered per port the size of the HSN packets can be configured for each port. The buffering of ports is done in a port object within the HSN.

On AAUSAT-II the HSN ports are used to allow high priority for some critical packets such as watchdogs and time dependent data for attitude control. However, for simplicity the low priority data packets, such as housekeeping, are transmitted on a port range that is specified for each transmitting subsystem (Source). A side effect is that the subsystems, when given an address on HSN, because of the CAN protocol also is given the priority they have within ports. Should two systems try to achieve a connection on the same port at the same time, the address of the subsystems defines who gets the connection.

The HSN utilizes both a Source and a Destination address, to define which subsystems are to respond on a transmitted packet. The Source and Destination fields are 6 bit addresses, and on AAUSAT-II there is currently only 6 addresses in use; this allows for several extensions on the network without changing the protocol. In the current configuration of the ports on AAUSAT-II, the ports alone allows for 256 different opened connections simultaneously on each subsystem. Should it be needed to introduce more than this the three last bits of the identifier bits are not yet used by HSN and the protocol could be expanded to yield these; see Figure 2.18 for location of the three bits.

Despite of the limited memory usage, the HSN protocol gives any subsystem on the bus a chance to establish a two-way communication with any other desired subsystem.

2.6.2 Packet Handling

The HSN protocol communications on the network are defined so that any established communication is sequential at a given port. This means that each transmitter that has completed a command or data transfer is given a status reply at the same port number as it was sent on.

As mentioned earlier only a message that is completed on the CAN is handled by HSN. This means that no errors on the data are detected by the checksum of the CAN controller. However, a risk of undetected errors still exists at a probability of less than $47 \cdot 10^{-12}$ [Bosch, 1991, p. 8]. Even this minor risk of a bit flips in the communication is critical to handle on a satellite. Should for instance a command sent to EPS be flipped before it arrives and the error is not detected, it could cause EPS to cut the power for something, these errors are hard to recreate and therefore difficult to detect on the ground.

Therefore, all communications on HSN is based on a Frame-Acknowledge relation which means that the transmitter (Source) sends only a single CAN frame until it receives an Acknowledge (ACK) from the destination system.

Only when an ACK is received by the transmitting subsystem, it will continue with the next frame in the data transmission. Should the receiver be handed a CAN message that even when acknowledged by the CAN controllers checksum still does not comply with the HSN protocol, the receiver will reply with an ABORT that will stop the transmission.

A CAN packet can be lost on the network, if no ACK or ABORT packets are replied to the transmitter. At least a single CAN controller acknowledges the packet on the CAN protocol level, but that unfortunately is not the CAN controller addressed by HSN, and none of the other CAN controllers have detected errors on the CAN level. HSN has implemented a timeout mechanism for waiting for an ACK response. When the timeout mechanism has passed, the transmitting subsystem will try to resend the lost frame. This procedure will be rerun until a maximum number of retransmissions have occurred, thus ensuring that the HSN packet will eventually reach the Destination if this is responsive on the CAN bus. The HSN protocol handles these retransmissions on a given port using the status of the #Q bit. Every time a new unique frame on a port is sent from Source this bit is flipped from its previous state. If this bit is not altered on a received HSN packet the receiver will interpret the packet as a retransmission. Only when this bit is flipped the data is added to the local buffer for the port.

To handle that the subsystems are working asynchronous and therefore could ask for access on a port already in use, a pending queue is created on the Destination. This queue holds the address of the sources that have sent a start packet on the port to gain access. If a timeout occurs at the Destination meaning that no transmissions or retransmissions has been detected for a given time, the pending queue is flushed, and all the sources in the queue are sent a NACK that will restart all the queued transmissions, thus a new pending queue will be created only containing subsystems that are still active. Remember that the transmitter on the queue that has the lowest Source address is the first one to gain access, if a connection is not already opened by a subsystem.

At AAUSAT-II the typical time before the maximum number of transmitting retries are reached is 1 second. This interval allows for a system to be unresponsive on the HSN network for almost a second, this should be considered when implementing the HSN protocol handling software. There exist plenty of available addresses on HSN without conflicting with the AAUSAT-II subsystems so the camera system could be given the first available address.

The camera system has been introduced and the tasks which should be handled by hardware and software have been specified. This has been supplemented by a description of HSN; the protocol used for communication at AAUSAT-II. In the following section requirements are listed and divided into functional requirements and requirements needed to fit the camera system into the satellite.

3 Camera System Requirement Specification

In this section requirements for the camera system are set up. These requirements are separated into a set of satellite requirements and functional requirements. Satellite requirements relate to demands which are necessary for being operational working in space and demands raised by the decision of following guidelines for AAUSAT-II subsystems. AAUSAT-II is used as reference since the mission for AAUSAT-III is not yet established.

The requirements marked with *(*)* are requirements that are taken into consideration to a certain degree in this project, but are not going to be tested in this project.

The requirements marked with *(**)* are requirements that are not taken into consideration and are not going to be tested in this project.

3.1 Satellite Requirements

Space environment makes physical demands to which the system must comply. The camera system is subordinated to the power and mass budget that, under normal circumstances, is set by the active project members working on the current CubeSat project. Since the planning of AAUSAT-III has not been started yet, those budgets have not been made and will not be completed before the end of this project. As a consequence, the project group has chosen to use some of the requirements from earlier AAU satellite projects, because it is likely that the requirements will be mostly the same for AAUSAT-III. In addition, requirements will exist for supply voltage and interface to communicate with OBC.

§SAT1 - ***(**) Mass budget of the camera system is set at*** **200 g.**
The requirement is set by the project group against a background of the mass budget of AAUSAT-II [AAUSAT-II, 2005d]. In the mass budget for AAU CubeSat the weight of the PL is specified as 125.9 g. Because AAUSAT-III will be three times larger in size, and is allowed to be three times larger in mass than both AAU CubeSat and AAUSAT-II, it is assumed that the allowed weight of the camera system can be higher than the weight of the camera system at AAU CubeSat. The exact size and mass of the total satellite is defined by California Polytechnic State University (Cal Poly) [University, 2004a]. Cal Poly is the launch coordinator and the managers of the CubeSat community; they have set the standard numbers for CubeSats.

§SAT2 - ***(*) The minimum operational temperature for the camera system must be*** **-40 °C** ***to*** **80 °C.**
This requirement was set by the satellite group at AAU working with AAUSAT-II [AAUSAT-II, 2005a]. The requirement is existent for all parts of the satellite. The large range is set because of the rough environment in space. The temperature range represents the worst case exterior temperatures. The interior temperature range is expected to be much smaller, due to the isolation provided by the hull of the satellite and the internal power dispatchment.

§SAT3 - ***(*) The camera system must be designed so that it is capable of operating at a pressure level down to*** **$5 \cdot 10^{-4}$ Torr $\approx$ 66.7 mPa** ***of vacuum.***
This requirement is set by Cal Poly and is present for all CubeSats [University, 2004b, p. 3].

§SAT4 - ***(**) The camera system must be able to operate in an environment with a radiation level of*** **10 krad.**
This requirement is set by the project group, because the radiation level is 3 to 10 krad according to Section 2.1.2, page 20, if a shield of 0.5 mm thick aluminum surrounds the electronic hardware.

§SAT5 - ***(**) The camera system must be built from parts that have an estimated lifetime of at least half a year. This means that the time it takes to outgas must be more than half a year.***
This requirement is set by the project group, because the estimated lifetime of AAUSAT-II is half a year and is expected to be about the same for AAUSAT-III.

§SAT6 - ***(*) The camera system must be able to withstand the vibrations during launch.***
The specific vibration pattern is set by Cal Poly [University, 2004b, p. 2].

§SAT7 - ***(*) The camera system must be assembled with a Class-3 soldering quality.***
This requirement is set by the AAU space program, and is a requirement for all systems launched into space, to ensure the durability of the system.

§SAT8 - ***(*) The camera system must be small enough to fit inside a*** **$10 \times 10 \times 10$ cm** ***CubeSat [University, 2004a]. It must be able to fit on an AAUSAT-II subsystem PCB of*** **87×87 mm.** ***The lens system must have a maximum size of*** **$5 \times 5 \times 5$ cm.**
This requirement is set by the project group to comply with the project specifications. Even though AAUSAT-III is going to be three times larger than a $10 \times 10 \times 10$ cm CubeSat the camera system must be small enough to fit inside a small CubeSat, because it must be possible to use it on a later CubeSat as AAU is more engaged in the small CubeSats.

§SAT9 - ***The camera system has to use either*** **3.3 V** **($\pm$0.2 V)**, **5 V** **($\pm$0.2 V)** ***or both voltage levels.***
This is defined by the AAUSAT-II group and is used in this project because it will most likely be the same at AAUSAT-III [AAUSAT-II, 2005c, p. 16].

§SAT10 - ***Power budget of the camera system is set at*** **500 mW** ***total in active mode.***
This requirement is set by the project group against a background of the power budget of AAUSAT-II, where PL in the budget was set to use 459 mW [AAUSAT-II, 2004, p. 3]. Because AAUSAT-III will be three times larger than AAUSAT-II it is assumed, that the power budget will be a little less strict. It is also worth mentioning that the camera system is not going to be active at all time. When the camera system is not active it should not consume any power at all.

§SAT11 - ***The camera system must be able to communicate with the OBC by a HSN protocol via a CAN 2.0B bus.***
This requirement is set by the project group, because AAUSAT-II uses this type of internal communication and it is therefore likely that AAUSAT-III is going to use this too [AAUSAT-II, 2006].

3.2 Functional Requirements

In this section functional requirements for the camera system are specified.

§FUNC1 - ***(**) The optics should be able to capture a subject that is*** **100 km** ***wide*** **($\pm$30%)** ***onto the image sensor when the satellite is in LEO, at a distance of*** **600** ***to*** **900 km** ***from Earth.***
This requirement has been set in the early project phase by the project group. The camera system should be designed in such a way that the width of the subject, matching 100 km, will be achieved in the middle of the interval of 600 to 900 km, that is in a altitude of 750 km. If orbit happens in a lower or higher altitude a deviation will occur, which has been taken into account in connection with the stated $\pm$30% deviation. The selected width has been chosen because the group finds it realistic to be able to recognize the image content of this extract.

§FUNC2 - ***(*) The maximum pixel size is set at*** **100 m $\times$ 100 m.**
This requirement has been set in the early project phase. The requirement is only valid when the

camera is positioned in LEO.

§FUNC3 - ***The images captured by the camera system must be color images.***
This requirement has been set in the early project phase. It will be easier to recognize the subject if the captured image is a color image.

§FUNC4 - ***The time delay from a capture image command is received until the image is captured must be maximum 1 second when the camera system is in idle mode.***
This requirement is set by the project group. If it assumed that the command is instantly transferred by the HSN protocol, the delay of the camera system will determine how much the subject has changed since the command was relayed by CDH. If the delay, from a command is received to an image is capture, is too large, it will be impossible to capture the desired subject due to changes in attitude of the satellite.

§FUNC5 - ***The image, when compressed, must have a maximum size of* 256 kB*.***
This requirement is set by the project group against a background of the datarate budget for AAUSAT-II [AAUSAT-II, 2005b]. According to the budget the satellite will be capable of transmitting 25.2 kB of data per minute at full speed, this means that it will be possible to download an image in about 10 minutes if the total bandwidth is used. If the camera system has about the same datarate budget as PL of AAUSAT-II, one image corresponds to half the budget for one day. The link budget of AAUSAT-III is maybe going to be larger than that of AAUSAT-II, but because it is not for sure this camera system will be designed according to the link budget of AAUSAT-II.

§FUNC6 - ***The camera system must be able to divide the large image into chunks so that it can be downloaded in small parts. The image chunks must have a size of maximum* 10 kB*.***
This requirement is set by the project group. The size is chosen so that it is the same as that of a compressed thumbnail.

§FUNC7 - ***The camera system must be able to make compressed thumbnail images and the size of those must be no larger than* 10 kB*.***
This requirement is set by the project group. If the speed for AAUSAT-III is the same as for AAUSAT-II it will take about half a minute to download a thumbnail [AAUSAT-II, 2005b].

§FUNC8 - ***The non-volatile memory must be able to hold 5 compressed images and 5 thumbnails.***
This requirement is set by the project group. The project group has decided that it seems reasonable to store 5 images before having to delete any of them.

§FUNC9 - ***It must be possible to change the following camera settings; gain (light sensitivity) and duration of exposure.***
This requirement is set by the project group, because it is desirable to be able to improve the images, as an example if the conditions are different from what is expected.

§FUNC10 - ***The camera system must be capable of executing up to 20 commands sent right after another.***
This requirement is set by the project group, because it is desirable to be able to recieve multiple commands.

§FUNC11 - ***(**) Time of capturing images must be fetched from OBC and stored as date and time, down to seconds.***
This requirement is set by the project group. The reason why the time should be fetched from OBC is that it already keeps track of the time and therefore there is no reason for the camera system to have an internal clock too.

In the following section it is specified how the requirements without marks should be tested.

4 Test Specifications

The purpose of this section is to specify how all the unmarked requirements from the previous section should be tested. Test conditions will be specified separately for every requirement. For the marked requirements it will only be specified what should be tested and not how, because the test is not going to be carried out during this project. The same marks are used in this section as in the requirement specifications.

To improve the readability of the section the requirements are repeated before the test specification for the requirement is specified.

4.1 Test Description of Satellite Requirements

§SAT1 - *(**) Mass budget of the camera system is set at* 200 g.
Should be tested by putting the camera system on a scale and ensure that the weight does not go beyond 200 g.

§SAT2 - *(*) The minimum operational temperature for the camera system must be* -40 °C *to* 80 °C.
Should be tested by performing two separate tests with respectively the minimum and the maximum temperature given in the requirement. It should also be confirmed that the camera system can handle rapid changes in temperature in the specified temperature range.

§SAT3 - *(*) The camera system must be designed so that it is capable of operating at a pressure level down to* $5 \cdot 10^{-4}$ Torr $\approx$ 66.7 mPa *of vacuum.*
Should be tested using a vacuum facility to perform a test with a vacuum level of $5 \cdot 10^{-4}$ Torr. When the camera system has been fully integrated into the satellite the finished satellite should be tested according to the "Thermal Vacuum" test specifications from Cal Poly [University, 2004b, p. 3].

§SAT4 - *(*) The camera system must be able to operate in an environment with a radiation level of* 10 krad.
Should be tested using any kind of radiation source capable of delivering the amount of radiation specified in the requirement. The test should be carried out over a longer period of time.

§SAT5 - *(**) The camera system must be built from parts that have an estimated lifetime of at least half a year. This means that the time it takes to outgas must be more than half a year.*
Should be tested by prolonging the vacuum and thermal tests to a 6 month period.

§SAT6 - *(*) The camera system must be able to withstand the vibrations during launch.*
Should be tested according to the "Vibration" test specifications from Cal Poly when the satellite is fully assembled [University, 2004b, p. 2].

§SAT7 - *(*) The camera system must be assembled with a Class-3 soldering quality.*
Should be tested by verifying each soldering meets specifications of a Class-3 soldering through a microscope.

§SAT8 - *(*) The camera system must be small enough to fit inside a* $10 \times 10 \times 10$ cm *CubeSat [University, 2004a]. It must be able to fit on an AAUSAT-II subsystem PCB of* 87×87 mm. *The lens system must have a maximum size of* $5 \times 5 \times 5$ cm.
Should be tested by measuring the dimensions of the camera system and verifying that the PCB is no larger than 87×87 mm and the lens system has a maximum size of $5 \times 5 \times 5$ cm.

§SAT9 - *The camera system has to use either* 3.3 V (±0.2 V), 5 V (±0.2 V) *or both voltage levels.*
Should be tested by:

1. Connect the CAN interface of the camera system to a PC with an USB to CAN converter.
2. Power-up system on the chosen voltages.
3. Capture image.
4. Download image to a PC via CAN bus.
5. Verify the downloaded image by comparing it with the actual subject.

§SAT10 - *Power budget of the camera system is set at* 500 mW *total in active mode.*
Should be tested by:

1. Connect the CAN interface of the camera system to a PC with an USB to CAN converter.
2. Power-up system.
3. Capture image while measuring the joint power consumption.
4. Download image to a PC via CAN bus while measuring the joint power consumption.
5. Verify that the joint power consumption have not exceeded 500 mW.

§SAT11 - *The camera system must be able to communicate with the OBC by a HSN protocol via a CAN 2.0B bus.*
Should be tested by:

1. Connect the CAN interface of the camera system to a PC with an USB to CAN converter.
2. Power-up system.
3. Set the subsystems to send data to CDH.
4. Capture image.
5. Download image to a PC via CAN bus.
6. Verify the downloaded image by comparing it with the actual subject.

4.2 Test Description of Functional Requirements

§FUNC1 - *(**) The optics should be able to capture a subject that is* 100 km *wide* (±30%) *onto the image sensor when the satellite is in LEO, at a distance of* 600 - 900 km *from Earth.*
Should be tested by performing a downscaled test with all the numbers in the requirement divided with a factor of e.g. 10,000.

§FUNC2 - *(*) The maximum pixel size is set at* 100 m × 100 m.
Should be tested by performing a downscaled test, could e.g. use the same image as in **§FUNC1**.

§FUNC3 - *The images captured by the camera system must be color images.*
Should be tested by:

1. Power-up system.
2. Capture an image of a red subject.
3. Capture a second image of a green subject.

4. Capture a third image of a blue subject.
5. Download images to a PC via CAN bus.
6. Verify the subject of the first image is red, the second is green and the third is blue.

§FUNC4 - *The time delay from a take image command is received until the image is captured must be maximum 1 second.*
Should be tested by:

1. Power-up system.
2. Place a stop watch that can show time in milliseconds and is controlled from a computer in front of the camera.
3. Capture an image and make the microcomputer trigger the stop watch, when the Capture Image command is received.
4. Download the image to the PC via CAN bus.
5. Verify that the time shown by the stop watch on the image is below one second.

§FUNC5 - *The image, when compressed, must have a maximum size of* 256 kB*,*
Should be tested by counting the number of bytes used for storing the image captured in the test of **§FUNC4**. Verify that the number of bytes do not exceed 256 kB.

§FUNC6 - *The camera system must be able to divide the large image into chunks so that it can be downloaded in small parts. The image chunks must have a size of maximum* 10 kB*.*
Should be tested by:

1. Power-up system.
2. Download the first image chunk from the existing image from **§FUNC4** to a PC via CAN bus.
3. Verify the downloaded image chunk does not exceed 10 kB.

§FUNC7 - *The camera system must be able to make compressed thumbnail images and the size of those must be no larger than* 10 kB*.*
Should be tested by:

1. Power-up system.
2. Capture an image and include a request for a thumbnail.
3. Download the image to the PC via CAN bus.
4. Verify the downloaded image chunk does not exceed 10 kB.

§FUNC8 - *The non-volatile memory must be able to hold five compressed images and five thumbnails.*
Should be tested by letting the camera system take five images with thumbnails and making sure that the camera system does not run out of non-volatile memory. The test should be carried out by:

1. Power-up system.
2. Capture five images and include a request for thumbnail for each.
3. Download all images and all thumbnails to a PC via CAN bus.

4. Verify the downloaded images by comparing them with the actual subject.

§FUNC9 - *It must be possible to change the following camera settings; gain (light sensitivity) and duration of exposure.*
Should be tested by:

1. Power-up system.
2. Capture an image.
3. Change camera settings so the exposure time of the camera system is decreased.
4. Capture a second image.
5. Download both images to a PC via CAN bus.
6. Verify the effect of the camera settings by comparing the downloaded images with each other that the second image is darker than the first.

§FUNC10 - *The camera system must be capable of executing up to 20 commands sent right after another.*
Should be tested by:

1. Power-up system.
2. Send a capture image command.
3. Send a list image command.
4. Repeat item two and three four times.
5. Send a delete image command.
6. Send a list image.
7. Repeat item five and six four times.
8. Verify that the number of images increases in the first five lists and decreases in the last five lists.

§FUNC11 - *(**) Time of capturing images must be fetched from OBC and stored as date and time, down to seconds.*
Should be tested by:

1. Power-up system.
2. Retrieve time from CDH immediately followed by next step.
3. Capture an image.
4. Retrieve time of capturing.
5. Compare the retrieved time from CDH and time of capturing.
6. Verify if the time difference does not exceed 1 second.

5 System Description

5.1 Choice of Components

This section describes the choice of components for the camera system. These components are:

- Image Sensor.
- Microcontroller.
- External Memory.
- Control Logic.

Before choosing each component, general considerations for all components are discussed to provide overall guidelines.

5.1.1 Overall Component Considerations

The environment in space is rough and components must be chosen carefully. Even though the components are protected by the MECH they must still be able to withstand vacuum, radiation and a large temperature range.

Size and Mass

The available space inside the satellite is limited and the camera system must utilize components of small size to achieve **§SAT8**. Components offering a large degree of integrated functionality are favored, to reduce the amount of components. This will also make it easier to keep the mass budget as required by **§SAT1**.

Voltage Levels and Power Consumption

The camera system has to use either 3.3 V, 5 V or both voltage levels according to **§SAT9**, page 42. If components operate at a 3.3 V supply voltage, it will, in generally for CMOS ICs, result in a smaller power consumption [Burd, 1995]. If it is not possible to get parts compatible using 3.3 V, then components using 5 V can be used. Components using lower voltages than 3.3 V can, if necessary, be implemented using voltage regulators. However, they are considered undesirable, since digital logics running at lower voltages have a smaller noise margin on inputs and are thereby less immune to noise [Ott, 1988, p. 276]. The 3.3 V voltage level is recommended for AAUSAT projects as compromise between radiation sensitivity and power consumption [03gr731, 2003, p. 13].

Radiation Sensitivity

Radiation-hardened components are expensive and might be difficult to obtain. However, devices using CMOS technology are generally not too sensitive to radiation. If it is not possible to obtain devices, which are guaranteed insensitive to the specified radiation level, components should use CMOS technology. To reduce the consequence of radiation the components can be shielded as explained in Section 2.1.2, page 20. This will, however, make it difficult to keep the mass budget. Since shielding is not an option for all subsystems on the satellite, it cannot be expected to be an acceptable solution.

Package Material

In Section 2.1 on page 19 it was determined that components should be able to withstand vacuum in space. Components should be housed in packages of non-plastic materials, for instance ceramic. If components use plastic packaging their lifetime in vacuum will be limited unless they are treated with Solithane 113300 as described in Section 2.1.2 on page 20. In this first prototype of the camera system plastic packaging without Solithane 113300 treatment can be used, since it is not vitally important for the electrical implementation what housing components are in.

Pin Package

To make sure all components can be connected without the use of special assembling processes, all ICs must use pins or edge connectors. This means that packages using covered surface mounts like BGA must be avoided.

5.1.2 Image Sensor

It has been decided to narrow the choice of image sensor into devices using CMOS technology, due to the advantages described in Section 2.3.3, page 30. The most important advantages are: Higher immunity to radiation, lower power consumption and integrated circuitry with converters and controllers. The use of an integrated solution will allow easier integration with the microcomputer of the system.

A very limited number of image sensors were obtainable through suppliers/sub-suppliers. By trying to contact manufactures directly it was discovered that they prefer selling large quantities and do not always make technical data public due to the competition in this field. Through a co-operation with the camera system company Devitech ApS, it was possible to choose among image sensors from a larger number of manufacturers which they have contacts to.

Obtainable Image Sensors

The image sensors which where found most suitable among the obtainable, where identified and their technical specifications are presented in Table 5.1, among these the final choice of image sensor was made.

Manufacturer	**Kodak**	**Cypress**	**Cypress**	**Micron**	**Micron**
Model	KAC-9648	Star 1000	CYIWOS-C3000AA	MT9T001	MT9D011
Resolution	1288×1032	1024×1024	2048×1532	2048×1536	1632×1216
Radiation	N/A	230 krad	N/A	N/A	N/A
Photoelectric sensitivity	2.5 $\frac{V}{lux \cdot s}$	N/A $\frac{V}{lux \cdot s}$	> 1.0 $\frac{V}{lux \cdot s}$	> 1.0 $\frac{V}{lux \cdot s}$	1.0 $\frac{V}{lux \cdot s}$
Sensitive area	1/2 ”	1 ”	1/2.8 ”	1/2 ”	1/3 ”
Color filter	RGB Bayer	No, B/W	RGB Bayer	RGB Bayer	RGB Bayer
Pin package	48 LCC	72 J-Leaded	48 PLCC	48 PLCC	Die or Wafer
Storage temperature	-40 °C to +125 °C	-10 °C to +60 °C	-30 °C to +85 °C	-40 °C to +125 °C	-40 °C to +125 °C
Operational temperature	-10 °C to +55 °C	0 °C to +60 °C	-30 °C to +70 °C	0 °C to +60 °C	- 30 °C to +70 °C
Supply voltage(s)	3 V	5 V	1.8 & 2.8 V	3.3 V	1.8 & 2.8 V
Power consumption	150 mW	$<$ 350 mW	215 mW	240 mW	77 mW

Table 5.1: *Comparison table of specifications on the obtainable image sensors, which are considered best suited for the camera system.*

Cypress Star 1000 is the only image sensor directly targeted for use in space and other environments with heavy radiation, and guarantees to be operational at the required radiation amount,

specified by **§SAT3**. However, since it is not available with a color filter array it is unable to capture color images as required by **§FUNC3**. Apart from this it also features the highest power consumption among the obtainable image sensors and have a very narrow thermal storage range. The narrow storage thermal range is a problem, because it can be difficult to ensure this during transport and satellite launch.

Cypress CYIWOSC3000AA and Micron MT9D011 are both image sensors targeted for use in cellular phones and features a very wide thermal operating and storage range. However, they both require a power supply of both 1.8 V and 2.5 V, which makes them troublesome to integrate into the camera system.

Micron MT9T001 and Kodak KAC-9648 are both image sensors targeted for use in digital still or digital video cameras and feature a wide thermal storage and moderate operating range. These image sensors will as so not be damaged even if large temperature variations occur, but it must be ensured, by OBC, that these image sensors are within the required thermal range before they are turned on.

The Chosen Image Sensor

Micron MT9T001 has been chosen as the image sensor to use in this project. Apart from consideration of temperature and operating voltage, the image sensor is used by Devitech ApS. This means that it will be possible to ask Devitech ApS for guidance on how to operate the image sensor. A sponsor agreement has been engaged with Devitech ApS. Devitech ApS will supply image sensors to the project for free, in exchange for having printed their logo on public released images captured with the system.

Limiting the Amount of Image Data

The image sensor has an array of 2048×1536 sensitive pixels, but only 1000×1000 pixels are required as a consequence of **§FUNC1** and **§FUNC2**. To reduce the amount of raw data transferred from the image sensor, it is decided to make use of the image sensors windowing or skip functionality. The windowing function allows the image sensor to crop out a desired piece of the entire sensitive area and only transfer this data. The skip makes it possible to divide the amount of pixels readout by an integer, by skipping $\frac{1}{2}$ or $\frac{2}{3}$ of the pixels readout on horizontal and/or vertical length and thereby reducing the amount of transferred pixels. The image data, which the skipping operation are performed on, are the pixels selected with the window function.

For prototyping the camera system it has been decided to use the 2× column skip function and limit the image data to 1024×1536 pixels. This solution will, however, result in an image which looks horizontally squeezed. If another resolution is suitable for a final lens system, this setting can easily be changed.

To ease an effective data storage it has been decided to only extract the 8 MSB of each pixel. Capturing all 10 bits would allow higher dynamic range and possibly produce a better image. However, data storing operations are performed on addresses of either bytes or half words. This means that only 10 out of 16 bits would be used and the remaining 6 bits would be unused, meaning 37.5% of the data storage would be wasted. Otherwise it will be needed to spread the bits of some of the pixels. If this method was used the first half word in the memory would contain all 10 bits of the first pixel and the 6 first bits from the second pixel. The following half word in the memory would contain the remaining 4 bits of the second pixel and all the 10 bits of the third pixel but also the first 2 bits of the fourth pixel. This is illustrated on Figure 5.1.

This solution would, however, complicate both the storing and loading operations of the image data, since two memory accesses are needed to obtain the data of several of the pixels, as it was the case with the second and fourth pixel in the explanation. For the microcontroller this might only be a question of writing extra functions, but for a DMA controller it would require latching the data bus instead of just rerouting it with the address bus.

Module for Prototyping

Devitech ApS produces module based systems. One of these modules is a BlackEye Platform PCB containing a Micron MT9T001 supplied with temperature measurement sensor, voltage stabilization, output buffer and external pin connectors. It has been decided to make use of the offer from

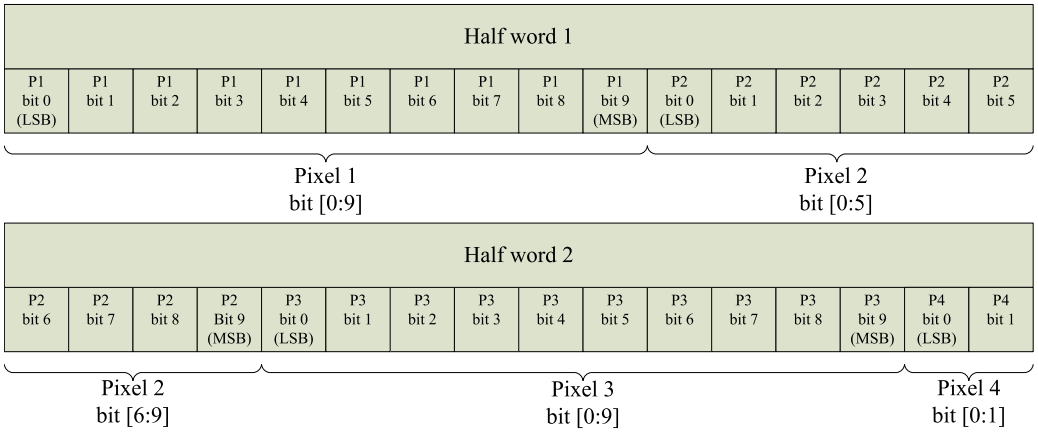


Figure 5.1: *Some halfwords contains data from up to three different pixels.*

Devitech ApS of supplying this module along with a schematic, instead of supplying the detached image sensor. In addition to the image sensor module itself, Devitech ApS supplies an optical system which does not match the optical requirement, but fits onto the image sensor module and will be suitable for testing the functionality of a prototype on Earth. This makes their module very suitable for prototypes of the camera system, because time saved on wiring and ensuring the operation of the image sensor can be spent debugging the microcomputer. It should, however, be noted that this module uses 5 V power supply due to the stabilization. The schematic is not included in this report due to the Non Disclosure Agreement.

Along with the assembled image sensor module, Devitech ApS also supplies the module as disassembled parts and a ready to solder PCB. For a final model of the system, it will be possible to make a custom PCB layout and/or ensure that the quality of the solderings meets the requirements for space as specified by **§SAT7**. A single component is a BGA component. So equipment must be found for soldering this component, or a replacement should be found for a new PCB.

5.1.3 Microcontroller

The microcontroller should possess enough processing power to carry out the needed image processing, and still it must feature low power consumption. It is also necessary for the microcontroller to address a sufficient amount of external memory since image processing requires a considerable amount of working memory. Built-in peripheral controllers are considered an advantage since they allow easy integration of peripheral devices and reduce the amount of external components. If at all possible the microcontroller should have a CAN 2.0B bus interface implemented. The amount of power consumed by unused functionalities should, however, be kept in mind.

In the case where memory access and allocation of data are carried out by the microcontroller, the speed gained at a Harvard architecture is appropriate during time of exposure. Thus, if a Von Neumann structure is selected, the data from the image sensor could perhaps be delivered faster than the microcontroller can handle it. In this case Direct Memory Access (DMA) needs to be incorporated to handle the allocation of data.

Besides, the technical specifications, support is considered important for the choice of microcontroller; this is crucial for first time users as many mistakes can be avoided or at least discovered. The issue of support narrows the actual choice down to two processors using either the Motorola 68000 or the ARM7. Both are based on the Von Neumann structure, thus regardless of the choice it will be necessarily to implement external DMA.

The Motorola 68000 is well on in years and does not match the speed nor the extra functionality offered by modern ARM7 microcontrollers. The Motorola 68000 is rated to 2 MIPS at 20 MHz [Freescale, 2006] and the lack of speed would increase time spent on image processing. Moreover, the the lack of extra functionality involves that it will require extra work to implement required functionality. In the following a microcontroller based on the ARM7 core is investigated with reference to examine its qualifications.

Atmel AT91SAM7A1

The Atmel AT91 series is a range of low-power 32-bit RISC microcontroller utilizing ARM cores. The AT91 series contains several models with a different amount of build-in functionality. The AT91SAM7A1, from this series, is found very suitable; it features [Atmel, 2005, p. 1].

- CAN controller with support for both 2.0A and 2.0B.
- Up to 16 MB of addressable memory using six chip select lines for configurable address decoding.
- Clock generators for both internal operation and output to external devices.
- Serial peripheral interface that allow serial communication to be memory mapped.
- 49 Programmable input/output lines.
- Up to 36 MIPS at 40 MHz.
- Internal 4 kB SRAM with rapid access, suitable for storing essential functions.
- Operating temperature range from -40°C to +85°C.
- 3.3 V supply voltage.
- Available in a 144-lead LQFP package and measures only 20×20 mm.

AT91SAM7A1 uses the ARM7TDMI core, which provides low power consumption in proportion to its performance. In Table 5.2 a comparison of the ARM family is shown. The ARM7 distinguish itself by using Von Neumann structure and having the lowest power consumption of the ARM processors.

	ARM7	ARM9	ARM10	ARM11
Typical MHz	80	150	260	335
mW/MHz	0.06	0.19	0.5	0.4
MIPS/MHz	0.97	1.1	1.3	1.2
MIPS/mW	16.6	5.8	2.6	3.0
(Dhrystone VAX MIPS)				
Architecture	Von Neumann	Harvard	Harvard	Harvard

Table 5.2: *ARM family attribute comparison [Sloss et al., 2004, p. 40].*

An Atmel AT91SAM7A1 will be used for the camera system according to the above description. Another important reason is that the AAUSAT team possess expert knowledge concerning this processor, since it is used for the OBC on the AAUSAT-II project [AAUSAT-II, 2005e]. Not only will it be possible to support from the AAUSAT team, but it will also be easier for the AAUSAT team to use and modify the system.

It is chosen to run the microcontroller at the maximum core clock of 40 MHz due to the wish of maximum processing power during image processing. The microcontroller always has a quiescent current, thus the power consumption vs. core clock frequency is improved at higher clock frequencies.

The ARM Architecture

The ARM core uses RISC architecture, i.e. a design philosophy aimed at delivering simple but powerful instructions at a high clock speed. The key point in a RISC design philosophy is to improve performance by reducing the complexity of instructions [Sloss et al., 2004, p. 15]. This approach often requires one operation being translated into more instructions, and thereby it depends more on the translation of the compiler [Sloss et al., 2004, p. 4]. RISC processors have a large general-purpose register, thus any register can contain either data or an address. ARM uses a modified RISC design philosophy that targets good code density and low power consumption. Besides, offering traditional RISC instructions, the ARM core also possesses some complex instructions that

take several clock cycles to execute. Thus, the ARM core is not a pure RISC architecture and in some sense, strength of the ARM core is that it does not take the RISC concept too far [Sloss et al., 2004, p. 5]. The ARM7 has a three-stage pipeline [Sloss et al., 2004, p. 30]. A pipeline speeds up execution by fetching the next instruction while other instructions are being decoded and executed. The steps in the ARM7 pipeline are shown below:

- Fetch
  Loads an instruction from memory.
- Decode
  Identifies the instruction to be executed.
- Execute
  Processes the instruction and writes the result back to a register.

5.1.4 Short Term Storage

With the chosen settings, the image sensor transfers 1.5 Mpixels of 8 bits each. To keep data temporary and process data from the image sensor a considerable amount of RAM is needed. Just keeping image data temporary requires at least 4.5 MB. With 1.5 Mpixel, 8 bit per color channel, and 3 channels in all the size is 4.5 MB.

For AAUSAT-II one module of Cypress CY62167DV30 16 Mb is being used. This low power CMOS static RAM works at a 3.3 V supply voltage [Cypress, 2005, p. 1]. Compared with dynamic RAM, static RAM has in generally better radiation performance [Faccio, 2000, p. 5]. Still, RAM is especially sensitive to radiation. CY62167DV30 and similar static RAM from Cypress offer a maximum capacity of 2 MB. Due to the above description it has been chosen to use three modules of Cypress CY62167DV30 resulting in 6 MB of available external RAM. The specific model used are the Cypress CY62167DV30LL-55ZI, where 55 refers to the fastest possible read or write cycle of 55 ns. The RAM module offers either byte or word data access. Depending on the mode data is either arranged as 2^{21} x 8 bit or 2^{20} x 16 bit.

5.1.5 Long Term Storage

In order to store compressed images when the camera system is powered down, it has been chosen to use FLASH. FLASH memory will also provide code storage. The advantage of this approach is that there will be no need for a PROM. Moreover, software updating could be possible just by transferring code to FLASH. However, since FLASH memory might be exposed to bit flips due to radiation, according to Section 2.1.1, page 19, protection is needed by means of shielding or code correction. The FLASH memory should have a capacity corresponding to 5 images of 256 kB, 5 thumbnails of 10 kB, 1 raw image of 1.5 MB plus code storage, resulting in a total of about 2.9 MB plus memory for code storage.

For AAUSAT-II two modules of AMD AM29LV320MT 32 Mb FLASH memory are being used. AM29LV320MT works at a 3.3 V supply voltage and has a low power consumption [AMD, 2005, p. 3]. The AM29LV320MT has been superseded by a successor, S29GL032M, but because AAU possesses expert knowledge concerning the AM29LV320MT due to the AAUSAT-II project, it has been chosen to use one module of AMD AM29LV320MT.

5.1.6 Control Logic

Data from the image sensor comes out in a stream. To transfer this data to RAM during capture image, it has been chosen to use DMA, because the the AT91SAM7A1 is using a Von Neumann structure. Thus, data can be transferred to RAM faster than the microcontroller is capable of controlling. The Control Logic should perform DMA and other logic functions.

Since PEELs typically have a maximum of 22 pins, it has been chosen to use a single FPGA rather than multiple PEELs to create the DMA. The XC2S30-5TQ144 FPGA from Xilinx supports 3.3 V input/output voltages and is chosen for this first revision of the camera system [XILINX, 2004, p. 9]. The reason for this is that XC2S30 has a suitable number of input/output pins and reads in its program from an external serial PROM at boot time, which is suitable proportional

to debugging and to introduce improvements in the program. Unfortunately, the Xilinx XC2S30 needs 2.5 V for internal logics, and a buck converter is needed to implement this FPGA. Another unfortunately consequence of the FPGA is that it possibly requires a large amount of power when it is powered-up and fetches its program [XILINX, 2001].

5.1.7 Summary of Chosen Components

In Table 5.3 the main components chosen for the camera system are summarized. In the table a "√" illustrates that a requirement is met, whereas a "÷" symbolizes that the requirement is not met. Power consumption is not evaluated for individual components as the power budget does not set up requirements for individual components but for the overall camera system. In addition, radiation sensitivity is neither evaluated; in fact this is difficult to set up because no radiation test results are available. The microcontroller, RAM and FLASH all meet the requirements concerning

Components	Temperature	Voltage(s)	Package Material	Operational Power
Image Sensor MT9T001	÷ 0 to +60 °C Only storage temperature is met	√ 3.3 V	√ 48-pin PLCC	– 240 mW
Micro-controller AT91SAM7A1	√ -40 to +85 °C	√ 3.3 V	√ 144-pin LQFP	– <300 mW @ 40 MHz <80 mW @ 8 MHz
RAM CY62167DV30 LL-55ZI	√ -40 to +85 °C	√ 3.3 V	√ 48-pin TSOPI	– 61 mW @ $f_{\max}$
FLASH AM29LV320 MT-120REI	√ -40 to +85 °C	√ 3.3 V	√ 48-pin TSOP	– 43 mW typ. active read power 165 mW typ. erase/program power
Control Logic XC2S30-5TQ144C	÷ 0 to +85 °C Industrial version offers –40 to +100 °C	÷ 2.5 V & 3.3 V	√ 144-pin TQFP	– Not specified (<500 mA current during power-on)

Table 5.3: *Summary of chosen components.*

temperature, voltage and package material. It is common for these components that they enter into the OBC of AAUSAT-II.

It appears from the Table 5.3 that the image sensor does not meet the requirements concerning the temperature. However, thermal design of the camera system and AAUSAT-III MECH may guarantee that the image sensor is within its operational temperature. Even so it should be checked if the image sensor is within operational temperature range before it is powered up.

The Xilinx XC2S30 FPGA is available in an industrial version, which meets the temperature requirements. The 2.5 V core voltage of the FPGA can be supplied by a buck converter or direct from EPS at AAUSAT-III, despite this is inexpedient if no other components of AAUSAT-III will operate at 2.5 V.

Image sensor, microcontroller, external memory and programmable logic have been chosen for the camera system. Thus, explanation of electric circuit design will be made in the next section.

5.2 Electric Circuit Design

The purpose of this section is to explain how and why the different components are connected to each other. To make the necessary connections, some small components will be needed, these will be described here too. To ease this descriptions, connections will be divided into groups of connections that has something in common. The full schematic of the circuit can be found on the Schematics on page 195.

5.2.1 Address and Data Bus

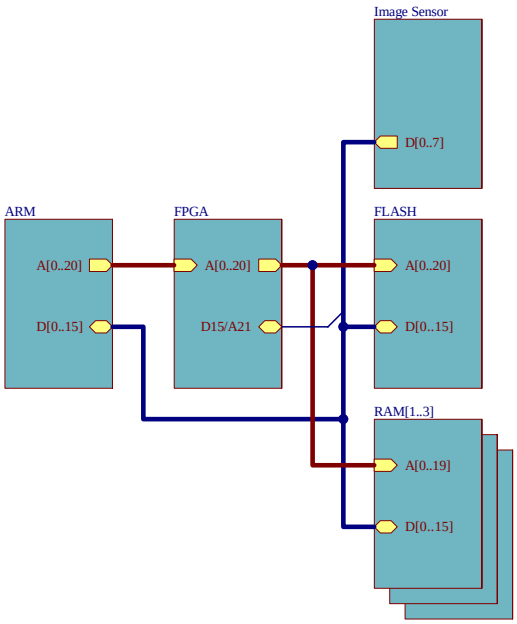


Figure 5.2: *Connection of the address and data buses.*

The address bus controls which memory address that is currently in use. The bus is 21 bits wide, making it capable of addressing up to 2 M addresses. As can be seen on Figure 5.2, the address bus is lead through the FPGA, which is done because the FPGA is handling DMA in the capture image process and the ARM is not capable of arbitrating the address bus in any other way than resetting the ARM. It is, however, undesirable to reset the ARM, because it will make it impossible to use a clock output from the ARM as clock signal to the image sensor.

When the bus is lead through the FPGA, resetting the ARM can be inhibited. In normal operation the FPGA simply replicates what the ARM puts on the address bus, but when the read-out of raw image data starts during the capture image process, the FPGA must disregard what the ARM puts on the bus and take the control of the bus itself to handle the DMA for the image sensor.

The data bus consists of 16 individual connections making the bus 16 bits wide. This bus is the main data transmission route in the system and transports data between the ARM and the two types of memory. The low byte, Data[0:7], of the data bus is connected to the image sensor. When an image is being captured, the FPGA controls the address bus to count up addresses in the third RAM module while the image sensor read-out the image data one pixel at a time to the data bus and into the third RAM module controlled by the FPGA. The image sensor has 10 bits of data outputs, but only the 8 most significant bits are used. This is done to fit one pixel into 8 bit of memory. Because the memory is accessed in 8 bit mode it takes 21 address bits to access the total RAM module and because of this data 15 becomes an address pin (A20) in 8 bit mode. 8 bit mode is also called byte mode.

5.2.2 Memory Control

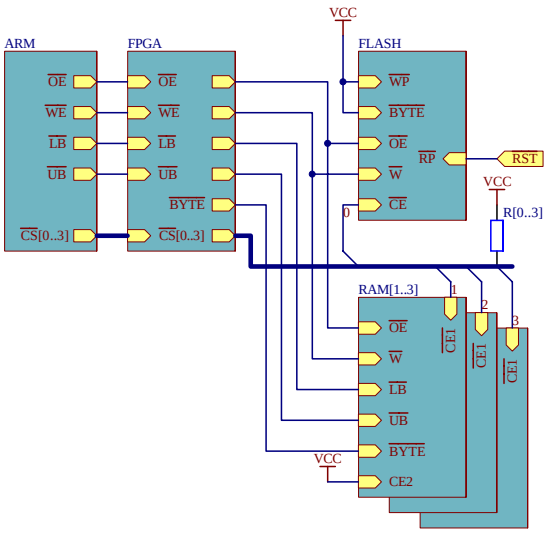


Figure 5.3: *Connection of the memory controls.*

To access the memory some control signals are needed. In normal mode the ARM needs to control the memory which means that the signals should go directly between those. Because the FPGA needs to gain control of the memory in the DMA process, signals from the ARM is connected to the FPGA, which passes through the signal in normal mode and control these signals in the capture image process. The signals controlled by either the ARM or the FPGA are Output Enable ($\overline{\text{OE}}$), Write Enable ($\overline{\text{WE}}$), Lower Byte ($\overline{\text{LB}}$), and Upper Byte ($\overline{\text{UB}}$). The Chip Select signals ($\overline{\text{CS}}$[0:3]) are also controlled by both ARM and FPGA, and thereby passed through the FPGA, but they also have a pull-up resistor to ensure that none of them are selected when the FPGA is being reset.

Byte mode is only needed on RAM in DMA mode and never on the FLASH. Byte mode is selected by pulling $\overline{\text{BYTE}}$ LOW. When operating in normal mode all addressing will be done in word mode. The $\overline{\text{BYTE}}$ pin on the FLASH is tied to V_{CC}, while it on the RAM is connected to the FPGA to allow it to put the RAM in byte mode. The Chip Enable 2 (CE2) pin on each RAM module is permanently tied to V_{CC} because only one chip enable pin on each block is needed.

There is no need to write protect a part of the FLASH so the Write Protect ($\overline{\text{WP}}$) pin is connected to V_{CC}. The Reset ($\overline{\text{RP}}$) pin is connected to the same reset signal as the ARM to reset them at the same conditions. The reset process will be explained later in this section.

5.2.3 Image Sensor Control

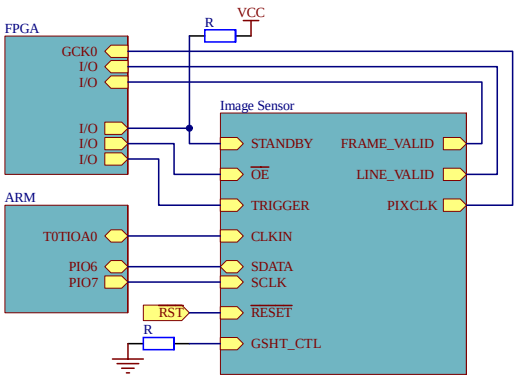


Figure 5.4: *Connection of the image sensor controls.*

The STANDBY pin of the image sensor is connected to the FPGA, which means that the FPGA controls the image sensor and if the ARM needs to access the image sensor, e.g. to send new configuration before the capture image process, it has to send a signal to the FPGA.

The pull-up resistor ensures that the image sensor is in standby mode when the FPGA is not active at boot time. To make sure the image sensor does not create a bus conflict, it must only put data on the data bus when an image is being read-out. To achieve this the Output Enable ($\overline{OE}$) is connected to the FPGA. By doing so, the FPGA can put the data pins into a high impedance state (Z) when not in the middle of the read-out sequence. To capture an image the TRIGGER pin on the image sensor has to be activated. Because the FPGA is handling this the pin is connected to the FPGA.

The image sensor needs a clock signal to operate. The clock generator of the ARM is utilized for this purpose. The ARM is set to deliver clock with a frequency of 10 MHz, which is a fourth of the core clock and the fastest frequency the ARM can output. The RAM is not guaranteed to operate faster than 18 MHz and the frequency of 10 MHz is found suitable [Cypress, 2005, p. 5]. The clock signal is outputted from the ARM at T0TIOA0 and should be activated before serial communication to the image sensor is started. The serial connection to the image sensor is needed to setup the image sensor. The serial connection consists of one data connection and one clock signal. The clock signal is produced by the ARM on the PIO7 pin and the data signal is bi-directional and connected to the PIO6 pin of the ARM. Because the PIO pins are used the serial interface is very flexible.

The $\overline{RESET}$ pin of the image sensor is connected to a different reset signal than the ARM, but resets under the same condition. Global Shutter Control is to be used in association with a mechanical shutter and the GSHT_CTL input is therefore not needed; thus it is tied to GND with a resistor.

IS-PIXCLK is the pixel clock output, which pulses every time a pixel is handled internally in the image sensor. This clock is connected to GCK0 on the FPGA to make them able of operating synchronously. The IS-FRAME is high during the entire frame including blanking pixels. The IS-LINE is only high as long as each pixels of the frame contains a valid pixel. Both connections from the image sensor are connected to the FPGA.

5.2.4 External Connectors

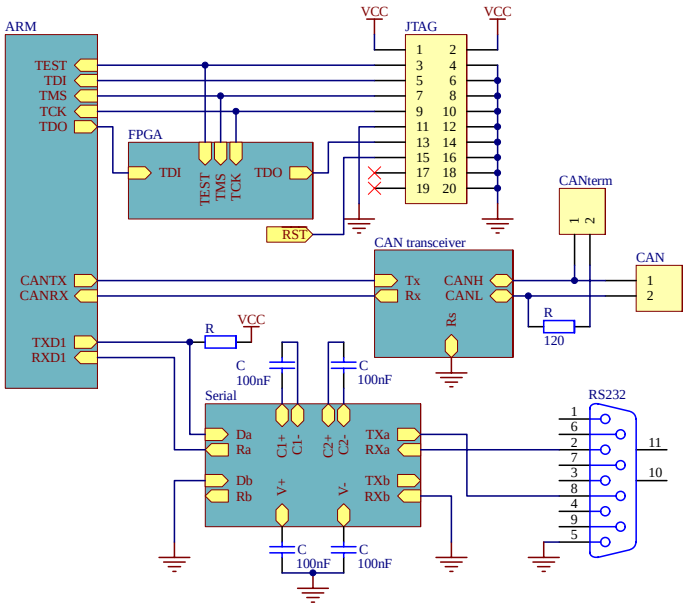


Figure 5.5: *Connection of the external connectors.*

Because this system is under development it is decided to keep debug options open. Both RS-232 and JTAG are connected to have both debug options. The JTAG is connected according to the

J-LINK specifications [SEGGER, 2005].

The AT91SAM7A1 microcontroller has an integrated CAN controller but lacks the electrical interface for the CAN bus. In relation to Appendix A.2 on page 141 concerning CAN, this means that the NRZ decoding and the protocol layer is covered by the internal controller of the ARM. Therefore, only the transceiver is needed to extend the ARM7 processor. For this task a SN65HVD230 is selected, which is a CAN transceiver that operates on 3.3 V and a bus voltage tolerance up to ±25 V. The Rs pin on the transceiver is tied to GND making it operate in high speed mode all the time because this mode is used at all times in the HSN protocol. A 120 Ω resistor is connected with a jumper between CANH and CANL to be able to choose whether the CAN bus should be terminated here or not.

To get the needed TTL levels on the RS-232 a transceiver is needed. For this task the MAX3232 is chosen, as it only needs 4 external capacitors. The capacitors are chosen to 100 nF as specified in the datasheet [Maxim, 1999, p. 9]. The input pins of the unneeded channel of the MAX3232 is tied to GND to avoid any floating pins. The pull-up resistor on the TXD1 pin avoids a floating pin when the ARM is reset at boot time.

5.2.5 Configuration and Control Pins for FPGA

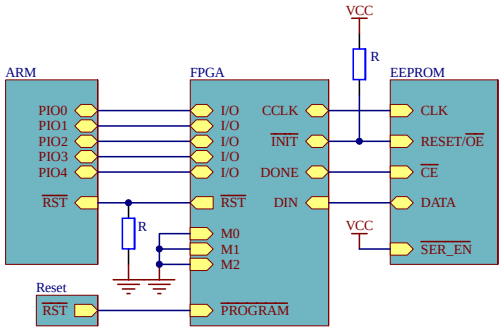


Figure 5.6: *Configuration and Control Pins to the FPGA.*

PIO0 to PIO4 from the ARM are connected to the FPGA to send control signals both ways; e.g. a signal is transferred from the ARM to the FPGA when an image should be captured, and the other way when the process has finished.

To prevent the FPGA from initialing its configuration process until the supply voltage has stabilized, a reset chip MAX6328 is added. The reset chip is connected to the $\overline{\text{PROGRAM}}$ pin on the FPGA. If $\overline{\text{PROGRAM}}$ is held LOW, the configuration of the FPGA is delayed. When the FPGA has been configured it negates the reset signals for other devices, which have been pulled down by external pull-down resistors during configuration of the FPGA.

Connecting M0 to M2 to GND on the FPGA ensures that the FPGA is configured as Master during the serial configuration of it. $\overline{\text{SER\_EN}}$ must be held High during FPGA loading operations. To configure the FPGA, which happens at boot time, an external ROM is used. For this purpose an AT17LV512 EEPROM is chosen. This is connected as specified in the datasheet [Atmel, 2006]. The EEPROM is chosen because it is simple to reprogram which is appreciated in the development phase. A pull-up resistor is added on $\overline{\text{INIT}}$ as this has an open-drain output; other pins require no pull-up for internal drop-in/stand-alone programming [XILINX, 2004, p. 72] [Atmel, 2004, p. 3].

To get the 2.5 V supply voltage needed for the FPGA internally a buck converter is chosen, namely a LM2619. The buck converter must be able of supplying the FPGA by a current of 500 mA caused by Power-On Requirements for the Spartan-II [XILINX, 2001, p. 1]. This is connected as specified in its datasheet [Semiconductors, 2003].

5.2.6 Summary of Connections

The chosen hardware components of the camera system has been connected to provide the functionalities required. The camera system connects the components on multiple busses. The busses have

different purposes and the design allows for the ARM and FPGA to handle all the data transfers. This can be seen on Figure 5.7 where an overview of all the data related connections are illustrated.

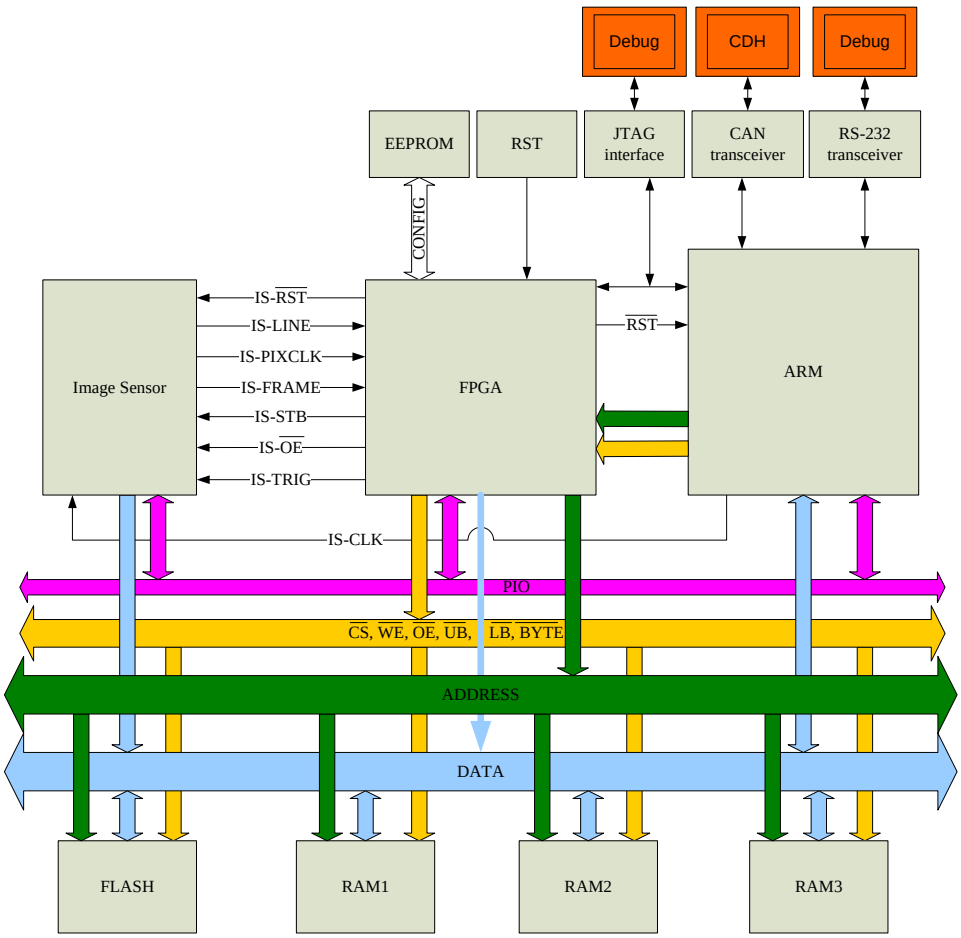


Figure 5.7: *An overview of the data and control busses in the camera system.*

The logic hardware is though not just connecting the dots, it is also a matter designing a PCB handling the production an thus reducing the noise as much as possible. In the following the designed circuit is realized.

5.3 Realizing the Electric Circuit Design

This section describes some challenges about making a PCB layout and will describe the actual PCB design. At first, component values not directly specified in datasheets will be calculated.

5.3.1 Determination of Component Values

The master clock and PLLRC pins on the ARM are connected as specified in the datasheet for the ARM [Atmel, 2005, p. 14]. The principle of a PLL and the set of external components it requires are explained and calculated in Appendix A.3, page 145.

Pull-up and pull-down resistors are calculated in Appendix A.4, page 147.

The component values determined in this report differs from the values shown on the circuit schematic in the case of pull-up resistors for chip select connections and values concerning clocks.

Due to unknown difficulties resulting in booting troubles for the first microcomputer system, component values identical to those used for the OBC of AAUSAT-II are used for the final prototype for the sake of safety. However, it is not expected that the difficulties were caused by the values of these components.

5.3.2 PCB Design

The purpose of this section is to explain why it has been chosen to use a PCB, which problem areas to be aware of when designing a PCB, and how the PCB is designed.

In this project it has been chosen to use a PCB because the chosen components are primarily Surface Mounted Devices (SMD) with many pins intended for PCB mounting. Besides, a PCB provides the opportunity to make a circuit that has EMC and therefore is more likely to work properly.

When designing digital circuits it is necessary to consider how the wires are routed, how long the wires can be, and how to decouple noise on the supply to make a functional circuit that has EMC and does not produce too much noise.

When designing digital logic circuits the internal noise is the primary concern [Ott, 1988, p. 276]. The four internal noise problems are

1. Ground bus noise.
2. Power bus noise.
3. Transmission line reflections.
4. Crosstalk.

These four items are examined in Appendix A.5, page 150.

Actual Design

To realize the PCB, GPV Printca A/S was contacted. They where willing to make a six layered PCB to a favorable price if their design requirements were met. The Design requirements that GPV Printca A/S sets for PCBs to satellites are defined by ESA. So the PCB is manufactured as space graded PCB in Glass Polyimide. Because the AAUSAT-II project needed PCBs too, a common order could be arranged and thereby it was possible to get a PCB.

The PCB is designed to have a lot of debug opportunities, because the PCB is delivered only a couple of days before the end of the project. The debug design makes it possible to separate the FPGA totally from the ARM by means of jumpers. This makes it possible to test the FPGA and the ARM separately. Furthermore, a JTAG and a RS-232 socket is mounted on the PCB for debugging opportunities.

The jumpers and sockets makes the dimensions of PCB larger than the dimension specified for the camera system in **§SAT8**. Here the requirement is $9 \times 5 \times 5$ cm for the camera system and the designed PCB is 7.0×13.8 cm. The PCB can be considerably smaller if some of the debug circuitry is removed.

The PCB is designed to follow the guidelines explained in Appendix A.5, page 150, when possible, to have EMC and decrease the internal noise.

To minimize the area between the signal conductors, GND, and V_{CC} the signal wires are placed between a GND and a V_{CC} plane. The GND and a V_{CC} planes also help to minimize the inductive and capacitive coupling between the different circuits on the PCB as shown in Equation (A.14).

Because of space restrictions no onboard decoupling capacitors are mounted where the power meets the PCB, but surface mounted decoupling capacitors are mounted near every IC to minimize ground bus noise and supply the current for current transients.

The PCB is designed to have wires shorter than the critical length. The longest and therefore most critical connection is the connection of A3+AC3, between the FPGA and RAM and FLASH, which is 0.271 m long.

To calculate the critical length of this connection the rise and transmission time has to be known. The rise time is calculated in Equation (5.2) with data from [Cypress, 2005, p. 4], [AMD, 2005, p. 55], and [XILINX, 2004, p. 44].

$$\tau_{\mathrm{r}} = \frac{C_{\mathrm{load}} \left(V_{\mathrm{OHmin}} - V_{\mathrm{OLmax}}\right)}{I_{\mathrm{out}}} \tag{5.1}$$

$$\tau_{\mathrm{r}} = \frac{(3 \cdot 8 + 6)\ \mathrm{pF} \cdot (2.4 - 0.4)\ \mathrm{V}}{12\ \mathrm{mA}}$$

$$\tau_{\mathrm{r}} = 5\ \mathrm{ns} \tag{5.2}$$

where:
τ_{r} is the rise time [s].
C_{load} is the load created by RAM and FLASH [F].
V_{OHmin} is the minimum HIGH output voltage [V].
V_{OLmax} is the maximum LOW output voltage [V].
I_{out} is the output drive current for the FPGA in the chosen configuration [A].

The critical length can be calculated as shown in Equation (A.17), page 151, [Bak-Jensen, 2006, p. 4], if the wave velocity is 138,000 km/s.

$$l = \frac{\tau_{\mathrm{r}}}{2\tau_{\mathrm{trans}}}$$

$$l = \frac{5\ \mathrm{ns}}{\frac{2}{138{,}000\ \mathrm{km/s}}}$$

$$l = 0.345\ \mathrm{m} \tag{5.3}$$

where:
v is the wave velocity [m/s].
l is the critical length [m].

The longest wire on the PCB is shorter than the critical length, which means that no termination is necessary.

The conductor width and spacing is designed to keep up with the demands setup in the ESA ECSS-Q-70-11A shown in Table 5.4, except for inner layer spacing. Here the design should have a spacing of 170 µm, but because the used PCB design program could not use different spacing on different layers, the inner layer spacing is set to 180 µm.

Layer	Design	Finished Board
Conductor width		
Inner layers	140 µm	120 µm
Outer layers	230 µm	200 µm
Spacing		
Inner layers	170 µm	150 µm
Outer layers	180 µm	150 µm

Table 5.4: *Required minimum conductor width and spacing for the designed PCB.*

The spacing and width between the conductors almost have the minimum dimensions written in Table 5.4. The spacing is not bigger, because of space restrictions, although the capacitive and inductive coupling and therefore crosstalk could be smaller as shown in Equation (A.20), Equation (A.22), and Equation (A.21) if the spacing was larger.

No inputs should be left open, because an input pin can pick up noise and make a gate switch randomly [Ott, 1988, p. 295]. Due to space constraints some input pins are left open, but to prevent the problem all unused PIO pins on the ARM is set to be output pins and set HIGH. Furthermore, a 150 Ω resistor is connected in series with the crystal, to minimize the harmonic current [Sondrup, 1995, p. 6].

PCB Modifications

Some modifications have been made to the PCB since it was sent to GPV Printca A/S. These modifications are entered in Table 5.5 together with some modifications that could be made to a future PCB with limited debug opportunities making space for other useful components. Furthermore, the PCB was originally designed to use a XC2S15-5TQ144C FPGA, but due to limits concerning the routing of the 21-bit counter in this model, this FPGA was later replaced by a XC2S30-5TQ144C FPGA. This FPGA is pin compatible with the XC2S15 but has a few more I/O pins, which makes the routing of the PCB illogical in some places. The full PCB layout can be found on the CD in the Schematics folder and on page 195.

Modification	Explanation
Modifications made to the PCB	
R_{121} is not connected.	External pull-up is only required for in-system programming.
R_{122} is not connected.	CCLK line does not require pull-up resistor.
$V_{CC,3.3}$ is connected to JMP1 pin 1.	$V_{CC,3.3}$ pulls $\overline{BYTE}$ HIGH in the minimum system.
Future modifications to the PCB	
Decoupling where the supply meets the PCB.	Minimizes conducted emission and susceptibility.

Table 5.5: *Modifications to the PCB.*

All hardware have been described and designed throughout the last sections. In the next section the requirements, functionality and implementation of the control logic performed by the FPGA are described.

5.4 Implementation of Logic in FPGA

The overall function of the FPGA and connection to other parts of the microcomputer have been explained. In this section the function and structure of the FPGA will be described in details, in order to program its internal logic circuitry. The FPGA requirements and premises are summed up; afterward a truth table for the output connections of the FPGA is presented. PIO connections used for communications between ARM and FPGA are assigned to specific functions and finally the Read-out sequence is specified. Afterward the program is designed and coded to meet a set of test specifications which are described in Section 8.1 on page 106. In the test section results of simulations and circuit tests are discussed.

5.4.1 FPGA Chip Architecture

A FPGA is a semiconductor device containing programmable logic components, which can be programmed to duplicate the functionality of basic logic gates or more complex combinatorial functions such as decoders or simple math functions. Even a microprocessor can be implemented in the larger and more advanced FPGAs. The internal architecture of a FPGA is shown on Figure 5.8. The logic is divided into a large number of configurable logic blocks (CLBs). They are distributed across the entire chip through programmable interconnections, and the entire array is surrounded by programmable I/O blocks (IOBs) [Wakerly, 2000, p. 882]. The CLB contains a set of simple logic building blocks, which are connected to internal memory holding the configuration of the FPGA. The specific contents of the logic building blocks depend on the FPGA model. A typical CLB consisting of a 4-input lookup table (LUT), a flip-flop, and a multiplexer as shown on Figure 5.9 [Wikipedia, 2006e]. The LUT makes the CLB capable of realizing all possible 4-input combinatorial logic circuits. From the CLB there is only a single output, which can be either the output latched

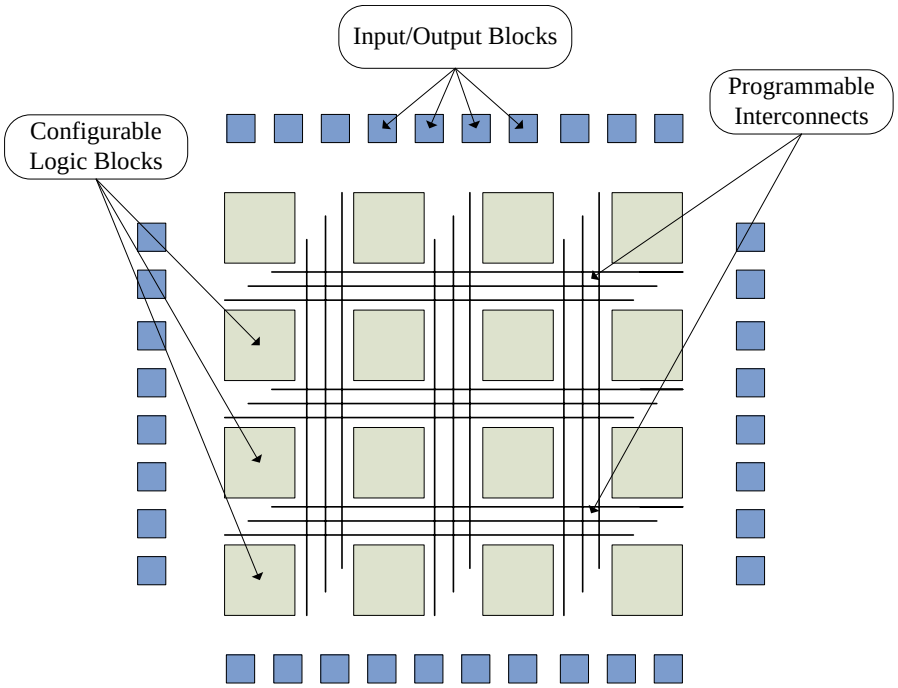


Figure 5.8: *Internal architecture of a general FPGA.*

in the register of the flip-flop or the present LUT output. Clock signals are typically routed via a dedicated routing network.

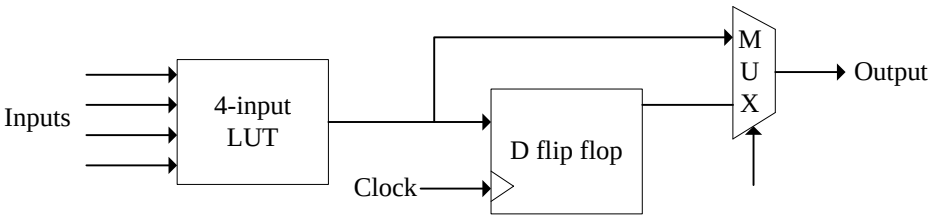


Figure 5.9: *Configurable Logic Block of a typical FPGA.*

5.4.2 Definition of Sequences

In Choice of Components Section 5.1, page 48, it is chosen that the FPGA must handle DMA control for the image sensor when it is triggered and starts to put raw image data on the data bus. However, this is only one of the sequences required to complete the Capture Image use case found in Section 2.5.1, page 34. Although, not all sequences implied in the Capture Image use case are yet defined, the sequences which are required to make it possible for the FPGA to perform the DMA control can be derived. These required sequences all include controlling the image sensor and some tasks of these sequences must be performed by the ARM. This means that software for the ARM must be written to comply with these sequences. These known sequences are carried out subsequent and they will be described in the following text.

1. Initialize image sensor.
   The sequence is used to make the image sensor ready to transfer raw image data before it is triggered. The image sensor must be enabled and settings must be transferred to its registers. Since details of this sequence rely on how the software for communication is coded the sequence is described later in Section 7.2 on page 91.

2. Read-out sequence.
   The purpose of this sequence is to trigger the image sensor and transfer an image of the subject to memory as raw image data. When the sequence starts, the FPGA relieves the ARM of address bus control and the FPGA handles the DMA control task for the image sensor. When the DMA control task is done and the raw image data stored, the FPGA returns the address bus control to the ARM. This sequence is described in details in Section 5.4.6.

3. Suspend image sensor.
   The sequence makes the image sensor power-down, since it is no longer used in the Capture Image process, and power can be saved. The steps of the sequence can be found in Section 7.4 on page 94.

Due to the switch of address bus control, two modes are defined for the FPGA:

- Pass-through mode.
  This mode lets signals from the ARM pass directly to the memory blocks.

- Read-out mode.
  The ARM is relieved of address bus control and the FPGA controls it for DMA purpose. This mode is enabled in the beginning of the Read-out sequence and disabled before the Read-out sequence ends.

5.4.3 FPGA Requirements and Premises

This section will sum up all previous requirements and premises set by decision made in the report, or derived from these. They are divided into different groups to provide a better overview.

- Overall.
  - The FPGA must handle DMA control for the image sensor during the Read-out sequence.
  - The FPGA must disable reset on both ARM and the image sensor, as soon as the FPGA is done loading its configuration, thereby allowing them to start-up. This should be done by negating the ARM-$\overline{\text{RST}}$ and IS-$\overline{\text{RST}}$ connections from their pulled down state.

- Interface to the ARM.
  - The communication signals between the ARM and the FPGA are sent through the five PIO connections.
  - The ARM must switch between Read-out mode and Pass-through mode with one of the five PIO connections.
  - The IS-STB connection must be controllable by one of the PIO pins in both Read-out mode and Pass-through mode. This is necessary since the ARM uses this connection during the initialization where the FPGA is in Pass-through mode.

- Pass-through.
  - Pass-through mode must be completely combinatorial since no clock input is available in this mode.
  - Byte mode must be disabled during Pass-through mode because the ARM performs 16-bit access on the external memory.
  - In Pass-through mode the FPGA must pass through the following signals: Address bus (Address[0:20]), $\overline{\text{CS}}$[0:3], $\overline{\text{OE}}$, $\overline{\text{WE}}$, $\overline{\text{LB}}$, $\overline{\text{UB}}$, and IS-STB

- Read-out.
  - Read-out mode will include sequential logic to handle DMA.
  - A 10 MHz clock input is available from IS-PIXCLK. It is enabled when IS-$\overline{\text{OE}}$ on the image sensor is asserted.

- Byte mode must be enabled during Read-out mode because the image data is stored as bytes instead of half words. This means RAM3 will operate with $2^{21} \times 8$ bit memory addressing (instead of $2^{20} \times 16$ bit) and an extra address connection replaces a data bus connection (Data 15).
- The raw image data should be stored in RAM3. Meaning $\overline{\text{CS}}3$ should be enabled in Read-out mode.
- The image sensor is initialized before the FPGA is used to perform the Read-out sequence. Thereby, the image sensor is ready to transfer raw image data when its output is enabled and its trigger pin is pulsed.

5.4.4 Assigning PIO Connections

Five connections have been established between the PIO pins on the ARM and the FPGA, making it possible for the ARM to control the FPGA and the image sensor that is connected to the FPGA. Each pin is dedicated to a specific function and its state corresponds to a specific function or a state of the circuit. Thereby, there is no need to decode more state connections to determine state of the circuit. This method is known as one-hot encoding [Golson, 1993]. This should ease understanding the description of the functionality and might help when tests are performed on the systems. In Table 5.6 it is shown what the PIO connections are assigned to. This assignment means that almost all control of the Read-out sequence is handled by the ARM. So, the functionality remain simple, and very limited control is performed by the FPGA. If desired, the number of connections between the ARM and the FPGA could be reduced. This could be done by letting the FPGA handle all of the communication with the image sensor during Read-out mode. If the communication between the ARM and the FPGA was mapped on to the address bus on an unused $\overline{\text{CS}}$, the need for dedicated connections could be decreased even further, or even be omitted. However, these solutions are rejected since both the ARM and FPGA feature plenty of connections. Such solutions would also complicate both the test and debugging of the system. If later revisions of the camera system need extra connections, this could easily be altered due to the programming flexibility of the FPGA.

Connection	LOW State	HIGH State
PIO0	Image sensor power on	Image sensor standby
PIO1	FPGA Pass-through mode	FPGA Read-out mode
PIO2	Frame invalid	Frame valid
PIO3	Does not reset FPGA counter	Resets FPGA counter
PIO4	Does not trigger image sensor	Triggers image sensor

Table 5.6: *The states of PIO connections correspond to specific functions or states. By using such assignments, the current operation of the FPGA can easily be determined.*

The input of PIO0 is passed through to IS-STB to control standby on the image sensor. PIO1 is the pin which the ARM uses to set the FPGA in either Pass-through mode or Read-out mode. The ARM can abort the Read-out sequence by setting PIO1 LOW to exit Read-out mode. PIO2 passes through IS-FRAME to the ARM. The ARM can thereby detect when all pixels have been transferred. PIO3 is used to reset the counter programmed into the FPGA. Before activating the transfer image data process, the ARM should pulse PIO3 to make sure the counter starts from zero. The input of PIO4 is assigned to pass through to IS-TRIG. After the counter has been reset the FPGA is ready and the ARM must pulse PIO4 to trigger the image sensor.

5.4.5 Truth Table of Output Values

Since all of the FPGA output connections have been assigned a truth table can be derived. The truth table presented in Table **??** shows the value of all output connections.

Output connection from the FPGA		**Output value from the FPGA**	
IC name	**Pin name**	**Pass-through mode**	**Read-out mode**
RAM/FLASH	Address[0-19]	ARM Address[1-20]	***Counting***
RAM	Address 20 / Data 15	*Floating (Z)*	***Counting***
FLASH	Address 20	ARM Address 21	*X (LOW chosen)*
FLASH	$\overline{\text{CE}}$	ARM $\overline{\text{CS}}$0	*Negated (HIGH)*
RAM1	$\overline{\text{CE}}$1	ARM $\overline{\text{CS}}$1	*Negated (HIGH)*
RAM2	$\overline{\text{CE}}$1	ARM $\overline{\text{CS}}$2	*Negated (HIGH)*
RAM3	$\overline{\text{CE}}$1	ARM $\overline{\text{CS}}$3	*Asserted (LOW)*
RAM/FLASH	$\overline{\text{OE}}$	ARM $\overline{\text{OE}}$	*Negated (HIGH)*
RAM/FLASH	$\overline{\text{WE}}$	ARM $\overline{\text{WE}}$	***Write pulse***
RAM/FLASH	$\overline{\text{LB}}$	ARM $\overline{\text{LB}}$	*Asserted (LOW)*
RAM/FLASH	$\overline{\text{UB}}$	ARM $\overline{\text{UB}}$	*Negated (HIGH)*
RAM	$\overline{\text{BYTE}}$	*Negated (HIGH)*	*Asserted (LOW)*
Image sensor	STB	PIO0	PIO0
Image sensor	$\overline{\text{OE}}$	*Negated (HIGH)*	*Asserted (LOW)*
Image sensor	TRIG	*Negated (LOW)*	PIO4
Image sensor	$\overline{\text{RST}}$	*Negated (HIGH)*	*Negated (HIGH)*
FLASH/ARM	$\overline{\text{RST}}$	*Negated (HIGH)*	*Negated (HIGH)*
ARM	PIO2	*Negated (LOW)*	IS-FRAME
X = Don't care, Z = High impedance, ***Counting*** = Used for counting address, ***Write pulse*** = Asserted when valid data is present and IS-PIXCLK is HIGH.			

Table 5.7: *Truth table of FPGA in Pass-through and Read-out mode. The two left columns list all the output connections and the two right columns shows their output value in respectively Pass-through mode and Read-out mode. All output values which are independent of an input are written with italic characters. It is shown what it is chosen to set don't care values to. The values written in normal characters represent inputs values which are passed through the output. The values written in italic bold are used for the connections handling DMA control.*

5.4.6 Specifying Read-out Sequence

The Read-out sequence is divided into seven steps and each step will be explained. As previously mentioned some steps are performed by the ARM and software must be written to follow these steps. The states of the FPGA connections during the steps of the Read-out sequence can be seen on the timing diagram presented on Figure 5.11.

1. **ARM asserts PIO1 to switch the FPGA into Read-out mode.**
   The switch to Read-out mode makes output values change on the FPGA. This switch involves enabling output on the image sensor and thereby enables its clock output. It must be made sure that the ARM does not perform any read or write operations after it has activated Read-out mode. If both the ARM and image sensor try to control the data bus, bus conflict will occur.

2. **ARM pulses PIO3 HIGH for at least 200 ns to reset the value of the counter in the FPGA.**
   The pulse on PIO3 will reset the counter inside the FPGA. The 200 ns is chosen because a period time is 100 ns at 10 MHz and two cycles are chosen to be certain.

3. **ARM pulses PIO4 HIGH for 500 ns to trigger the image sensor.**
   The pulse on PIO4 is forwarded to the trigger of the image sensor and the falling edge of the pulse triggers the image sensor causing it to start outputting pixel data. It was not possible to find exact requirements in the datasheet of the image sensor and Devitech ApS was contacted for guidance. Devitech ApS recommended five cycles and the chosen interval of 500 ns ensure that five IS-PIXCLK clock periods occur.

4. **ARM starts monitoring PIO2.**
   The ARM must verify that IS-FRAME is activated to confirm the image sensor has started to output pixel data.

5. **FPGA asserts $\overline{\text{WE}}$ when IS-PIXCLK and IS-LINE are HIGH.**
   The image sensor has started outputting pixel data on the data bus. However, until IS-LINE is HIGH only blanking pixels are written on the data bus. When both IS-PIXCLK and IS-LINE are HIGH, data should be written into the RAM by activating $\overline{\text{WE}}$. The exact timing constraints will be described later.

6. **FPGA increments counter-output on ⌝ IS-PIXCLK when IS-LINE is enabled.**
   The counter-output must increment before a new valid pixel is present on the data bus, thereby every new valid pixel is written to a new address. This is ensured by incrementing the counter on each falling edge of the IS-PIXCLK occurring simultaneously with IS-LINE. At the end of a line the IS-LINE is negated by the image sensor while invalid pixels used for blanking are put on the data bus. The counter is stopped while blanking pixels occur on the data bus and it does not increment until the valid pixels of the second line begins. The fifth and sixth steps are repeated until all pixels of the entire frame have been put out by the image sensor.

7. **When the ARM has detected that PIO2 has been negated, it negates PIO1 to switch the FPGA back to Pass-through mode.**
   When IS-FRAME is LOW the image sensor is either done saving the raw image to RAM3 or has not started DMA. As specified in the fourth step the ARM has already made sure that PIO2 has been asserted. Thereby, the ARM is sure that the image sensor is done saving the image when it detects that PIO2 is negated. Then the ARM can return the FPGA to Pass-through mode. This approach is done to make sure that the ARM does not exit Read-out mode when the image sensor starts up.

The pins PIO0 and IS-STB are not included on Figure 5.11 since their states do not change during the sequence. The PIO0 and IS-STB connections are set LOW in the initialization before entering Read-out mode and are kept LOW during the entire sequence. The IS-CLK clock output from the ARM is also enabled during initialization and keep running during the entire sequence.

5.4.7 Timing Requirements

To provide timing requirements for the FPGA, which will ensure proper read and write operations to and from memory, it is necessary to examine timing constraints set by the RAM and FLASH. The following text explains timing requirements in three different situations where delays of the FPGA influence.

Propagation Delays in Pass-through Mode

When the FPGA operates in Pass-through mode it will delay all of the memory control signals from the ARM, but not data signals since the data bus is not connected to the FPGA. However, the ARM is able to operate with memory of limited transfer rate by insertion of wait states in the memory operations. By adjusting the number of wait states the propagation delay can to a certain degree be compensated. For this reason, no requirements for a maximum propagation delay are set. Since the number of wait states must be kept as small as possible, it is still necessary to know how much propagation delay is added by the FPGA, and tests will be performed to measure the delay.

Time for Change Mode

When the ARM changes the mode during the Read-out sequence it changes the state of many control signals. It is necessary that the FPGA has completed the mode change before the ARM control signals can be expected to give the correct output. However, no requirement will be set since no-operation commands can be put into the software running on the ARM to make it wait for the FPGA to finish mode change. In order to find out how many no-operations commands are required the mode change delay must be estimated with simulations.

Timing the Write to Memory Pulse During the Read-out Sequence

The operation which writes pixel data to memory during the Read-out sequence will be further explained in this text. Since Figure 5.11 only provides an overview the exact timings with delays and constraints are shown on Figure 5.10 and values shown in Table 5.8. In connection with the write operation four constraints are found, it is described which constraints from RAM they relate to, and how their minimum and maximum tolerance have been found.

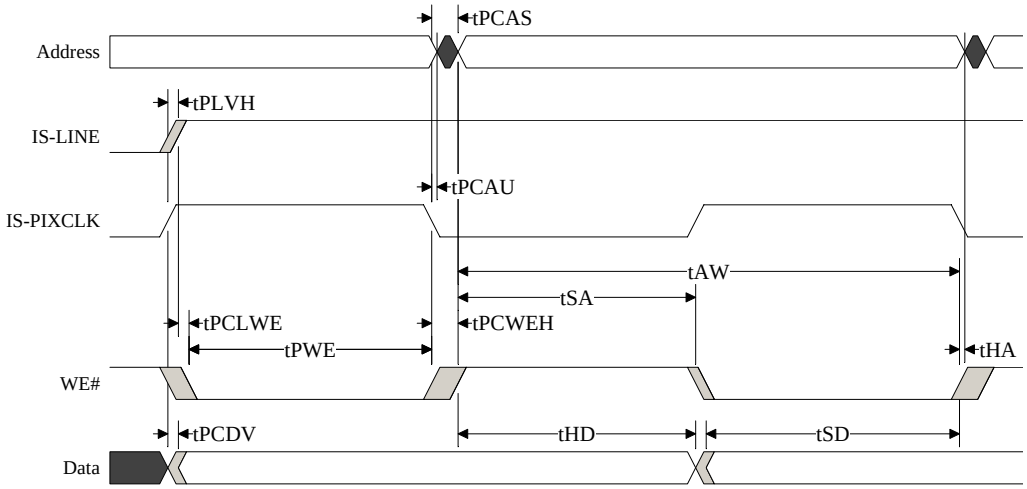


Figure 5.10: *Timing diagram of how the first two pixels of a line are written to RAM3 during the Read-out sequence. Timing constraints set by the RAM and delays of the image sensor and FPGA are shown. It is shown how IS-LINE is enabled when valid data is present on the data bus. When IS-LINE and IS-PIXCLK are HIGH the* $\overline{\text{WE}}$ *pin on RAM3 is asserted. The value on the address bus is kept until the falling edge of IS-PIXCLK occurs, simultaneously with an enabled IS-LINE. An explanation of the waveform states can be found in* Appendix A.8 *on page 161.*

IS-PIXCLK LOW to WriteEnable HIGH ($t_{\mathbf{PCLWE}}$)

The constraint t_{PWE} specifies that the write pulse is at least 40 ns. PIXCLK is a symmetric clock running at 10 MHz, meaning it is HIGH for 50 ns. For the first valid pixel in each line, the IS-LINE is delayed up to 2 ns (t_{PLVH}). This leaves a requirement for t_{PCLWE} of not being 10 ns - t_{PLVH} larger than t_{PCWEH}.

IS-PIXCLK & IS-LINE HIGH to Write Enable ($t_{\mathbf{PCWEH}}$)

The t_{HA} constraint requires that $\overline{\text{WE}}$ is HIGH before the address becomes unstable and increments. This constraint is critical because change of $\overline{\text{WE}}$ and address bus both happen at the falling edge of IS-PIXCLK. This constraint requires that t_{PCWEH} is smaller than the minimum t_{PCAU}.

Negative edge of IS-PIXCLK to Address Unstable ($t_{\mathbf{PCAU}}$)

As just specified the address must not become unstable before $\overline{\text{WE}}$ is negated. The minimum value is thereby set to $\mathbf{t}_{\text{PCWEH}}$, thereby the two requirements directly depend on each other. Using

Short	Name	Min	Max
Timing constraints from RAM			
t_{PWE}	PulseWidth write Enable	40 ns	-
t_{SD}	Setup Data (stable) to write end	25 ns	-
t_{HD}	Hold Data from write end	0 ns	-
t_{SA}	Setup Address (stable) to write start	0 ns	-
t_{AW}	Address setup (stable) to Write end	40 ns	-
t_{HA}	Hold Address from end write	0 ns	-
Image sensor delays			
t_{PLVH}	⎍ IS-PIXCLK to IS-LINE	0 ns	2 ns
t_{PCDV}	⎍ IS-PIXCLK to Data Valid	0 ns	2 ns
Constraints for FPGA delays			
t_{PCLWE}	IS-PIXCLK & IS-LINE HIGH to Write Enable	-15 ns + t_{PCDV} - t_{PLVH}	t_{PCWEH} + 10 ns - t_{PLVH} up to 60 ns - t_{PLVH}
t_{PCWEH}	IS-PIXCLK LOW to Write Enable HIGH	t_{PCLWE} + t_{PLVH} - 10 ns	t_{PCAU} up to 50 ns
t_{PCAU}	⎏ IS-PIXCLK to Address Unstable	t_{PCWEH}	50 ns + t_{PCLWE}
t_{PCAS}	⎏ IS-PIXCLK to Address Stable	t_{PCWEH}	50 ns + t_{PCLWE}

Table 5.8: *Timing constraints and delays, relevant to ensure proper timing of the write pulse, have been obtained from the datasheet of the RAM [Cypress, 2005, p. 7] and image sensor [Micron, 2005, p. 36]. From these a set of timing constraints to the FPGA is derived, which must be verified after the FPGA has been programmed. The highest and lowest tolerances of each derived constraint are explained in the text.*

simulation it will be estimated if the FPGA is able to keep this constraint. If simulation shows the constraint is kept the simulation is followed by circuit test to verify if the constraint is met. However, the t_{PCAU} must not be larger than allowing requirements to t_{PCAS} to be kept. The requirements for the maximum value are set equal to the t_{PCAS} requirement.

Negative edge of IS-PIXCLK to Address Stable (t_{PCAS})

The highest tolerable delay of t_{PCAS} is set by t_{SA}. It is necessary that the address is stable before the beginning of the write pulse, to ensure data on the previous address is not modified. Since the address shift on the falling edge and the write pulse switches active on the positive edge it leaves 50 ns + t_{PCLWE} delay at the positive edge as requirement for t_{PCAS}. The minimum value is set by requirements to t_{PCAU}, since the address cannot be altered and become stable before it has been unstable.

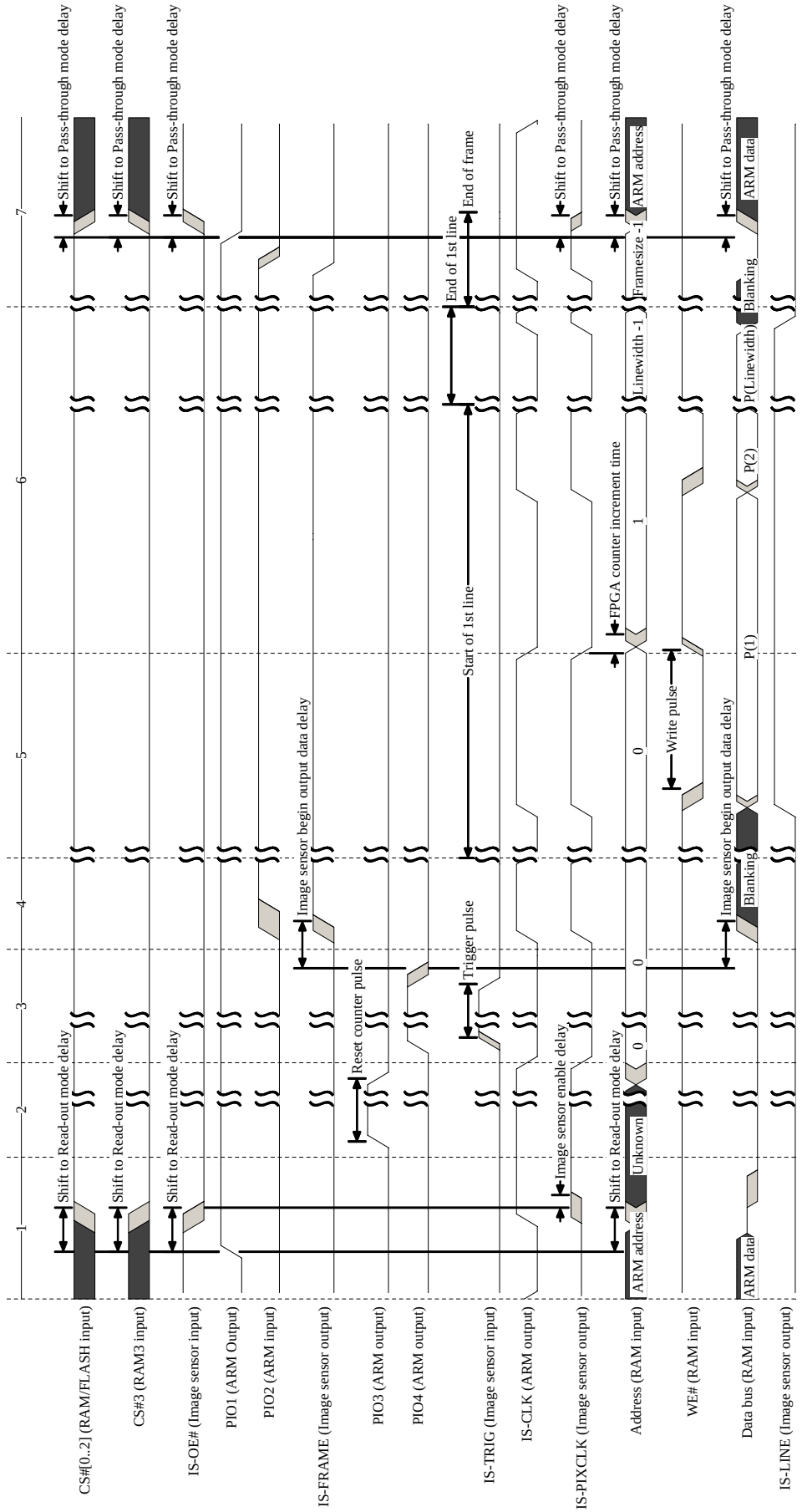


Figure 5.11: *Timing diagram of the entire Read-out sequence. The sequence is divided into seven steps corresponding to the described steps. It is recommended to examine the timing diagram and the text explaining it one step at a time. The diagram serves only as an overview and timings are not exact. An explanation of the waveform states can be found in* Appendix A.8 *on 161.*

5.4.8 Programming the FPGA

It is possible to write the program for the internal logic of a FPGA in a number of different ways. It would be possible to completely model the function of the FPGA graphically with a schematic using logical building blocks or gates. However, a verbal programming language is considered more flexible, since it is easier to change constants or alter a few lines of program code than to modify a circuit diagram by altering connection and adding extra gates. Just like ordinary programming the functionality is divided into smaller modules to provide an overview. The overview will ease separating functionality requiring combinatorial logic from functionality requiring sequential logic. By using a programming language it is still possible to import models of logical building blocks.

Introducing the VHDL Language

For programming the FPGA three well-known high level programming languages are available; these are ABEL, VHDL, and Verilog. Both Verilog and VHDL are simulation languages and only a small part of these languages are actually synthesizable in hardware, whereas ABEL is designed as a hardware description language only. VHDL is chosen for the programming of the FPGA.

VHDL stands for Very-High-Speed-Integrated-Circuit Hardware Description Language and can be used for design, simulation and synthesis of anything from simple combinatorial logic circuits to complete microprocessors [Wakerly, 2000, p. 264]. The VHDL standard is defined by IEEE and was originally developed for simulating digital circuits and could be used to document that a circuit would work as intended and meets requirements from time constraints [Chang, 1997, p. 1]. The very useful feature of synthesizing VHDL to actual circuits was not introduced until later.

Constructing a VHDL Program

A VHDL program is typically constructed from smaller modules each having an entity and architecture. The entity corresponds to the header file of a C program. It imports library packages and specifies which signals are to be used as input arguments for the module, and which signals that are to be returned from the module as output signals. The signals are specified as internal connections between modules (called buffers) or as a port for input, output or both [Wakerly, 2000, p. 270]. Libraries are typically used to import standard logic states such as: '1' (Logic 1), '0' (Logic 0), 'X' (Unknown) and 'Z' (High impedance), which are found in STD_ULOGIC [Wakerly, 2000, p. 273].

The architecture contains the actual functions of a module and these are specified with concurrent statements and processes. All concurrent statements are executed in parallel (simultaneous) in contrast to a C program, where code are executed one line by line (subsequent). [Wakerly, 2000, p. 282]. This happens because the VHDL describes digital logic and these statements are modeled by combinatorial circuitry. For a FPGA the modeling these statements are done through the memory setting up LUTs and multiplexers etc..

The processes are used to define subsequent logic and contains subsequent statements. The statements in a process are executed each time the process is run. To determine when the process must run the process contains a sensitivity list with a list of signals. When a signal in the sensitivity list change the process is run [Wakerly, 2000, p. 289].

VHDL uses two kind of data, variables and the previously mentioned signals. The signals are visible inside the modules which have declared these in their entity. Variables are only visible inside the process where they are declared. Variables may be made available outside the process by writing a statement, inside the process, which assigns the value of a variable to a signal. The variables are typically used as states for the subsequent logic of a process. This is possible since the values of variables can be latched in registers, as mentioned in the explanation of the internal FPGA architecture.

Writing VHDL for Synthesis

When programming a FPGA in VHDL it is important only to use VHDL constructs which are synthesizable. Non-synthesizable design will create no syntax failures, but report errors during implementation. Synthesis is not defined by the IEEE standard, thus different tools may have different capabilities. As an example a time expression like "wait for 20 ns", which are intended to simulate the delay of an input signal by 20 ns, is difficult to realize in hardware [Chang, 1997, p. 123].

It is important to remember that to synthesize a design this must correspond to actual hardware. For instance, using both rising and falling clock edges may not synthesize properly. The explanation for this lies in the way the hardware that should execute these functions are constructed; a flip-flop works only with one edge. Another example is omissions in process statements which will synthesis a latch if not all condition are defined [Chang, 1997, p. 128]. This happens since a variable will keep its value until it is assigned a new value. Therefore, it is important to always define default values when creating conditional assignments. According to ASIC design guidelines the safest method to keep control of timing is to create a design which is synchronous [ES2, 1993, App.1 p. 4]. A synchronous design implies that all state changes occur on the same edge of a single clock. If a local reset function for a part of the circuit is required, it is recommended to use one which is synchronous with the clock [ES2, 1993, App.1 p. 14].

Environment for VHDL Programming

Xilinx ISE 8.1 is chosen as compiler for the VHDL code because it contains a model of the chosen XC2S30 FPGA. It also provides the ability to compile the VHDL to a PROM image file. Such image file can be written into the EEPROM with a PROM programmer. When the EEPROM contains the image and is connected to the FPGA it will transfer the setup of the internal logic of the FPGA upon power-up.

Flow of the Design Method

To design the program for the FPGA, a design method was found with the following steps [Wakerly, 2000, p. 266]:

1. Hierarchy / Block diagram
2. Coding
3. Compilation
4. Simulation / Verification
5. Synthesis
6. Fitting / Place + Route
7. Timing Verification

This flow should be followed by transferring the VHDL to the serial EEPROM and by performing circuit verifications of the FPGA.

The steps will be carried out, but not all steps will be explained further. The block diagram is explained in the following text. The results of tests and simulations are discussed in the Section 8.1, on page 106. Source code for the VHDL program can be found on the CD in the folder Source Code for FPGA.

5.4.9 Description of FPGA Block Diagram

As shown on Figure 5.12 the functionality of the FPGA is divided into three smaller blocks, each handling a smaller task.

Counter Module

The counter module is a 21 bit up-counter, being used to increment the address on the RAM3 and thereby handling the address routing of the DMA control task. With 21 bits it is capable of counting up to address 2 M. To make the module count, a running clock input is provided from IS-PIXCLK and counting must be enabled with IS-LINE and PIO1. The counter can be reset if PIO3 is set HIGH synchronous with a clock. The counter has no stop value and it is thereby implemented as a loop counter. The counter is automatically stopped when the IS-FRAME is negated and a stop value would only make result in a less flexible solution. The counter module is sequential since its current output values are latched in registers, and a next value is incremented from this value.

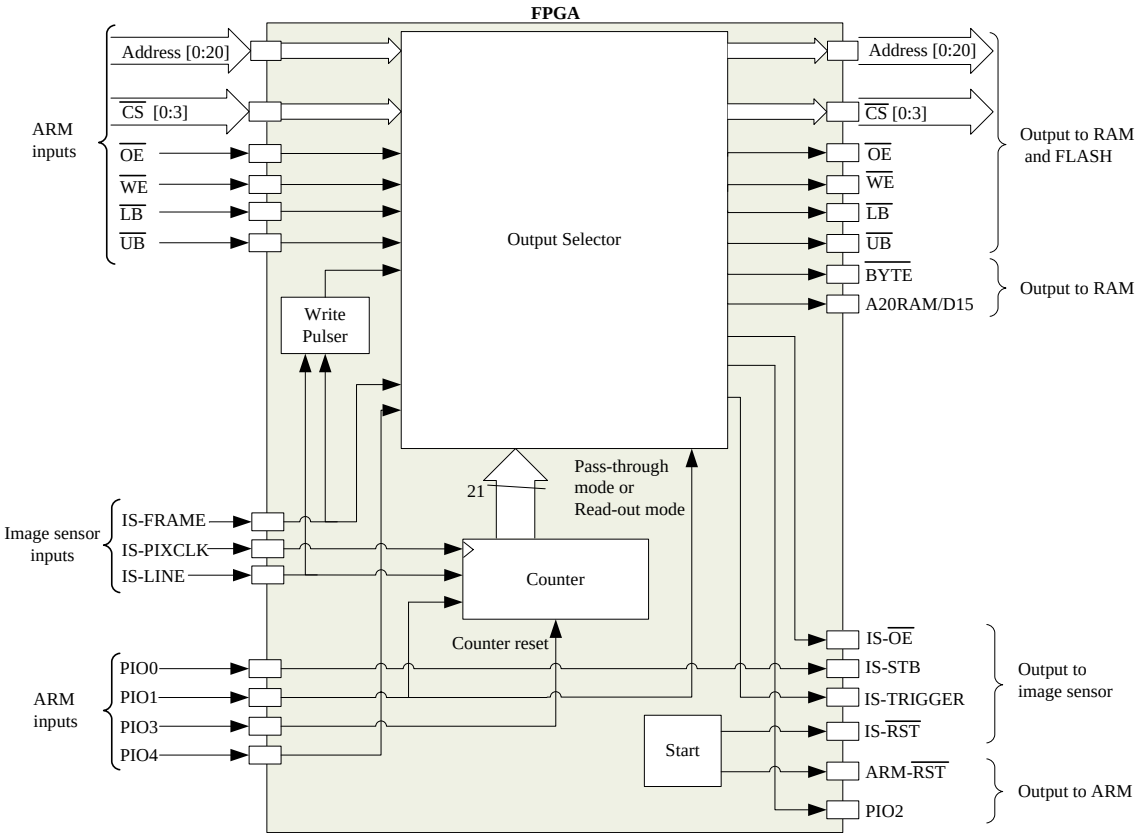


Figure 5.12: *Block diagram showing how the functionality of the FPGA is divided into modules.*

Write Pulser Module

This module generates the write pulse used in read-out mode. The module uses IS-LINE and IS-PIXCLK as inputs to assert $\overline{\text{WE}}$, only when both these inputs are asserted and valid data is created by the image sensor. Although IS-PIXCLK is a clock signal, the write pulser module will not use the edge, but only the state. Thereby, the module is completely combinatorial.

Output Selector Module

The output selector module is used to switch output signals for memory control between the values of Pass-through mode and Read-Out mode. The memory control signals from the ARM, the internal write pulser and counter modules are connected to the output selector module. The module is controlled by the PIO1 input. The function of this module corresponds to a multiplexer. In Pass-through mode the address and memory control output signals are simply passed from the ARM. In Read-out mode, the input from the counter module is passed as address output, the input from the write pulser is passed to write enable, and the remaining memory control signals are set to constant values. Besides memory control signals, the Read-out mode also enables the remaining communication between ARM and image sensor by passing the IS-FRAME to PIO2 and PIO4 to IS-TRIGGER. The module does not latch any values in registers and it is completely combinatorial.

Start-up and Constant Assignments

This module does not use any control signals as input, and only routes constant assignments and values which are independent of mode. The constant values deactivate the RESET on ARM, image sensor and FLASH. Since the outputs are set as high impedance until the FPGA has loaded its configuration from the EEPROM, the FPGA provides a power-up reset delay. This is shown on Figure 5.13.

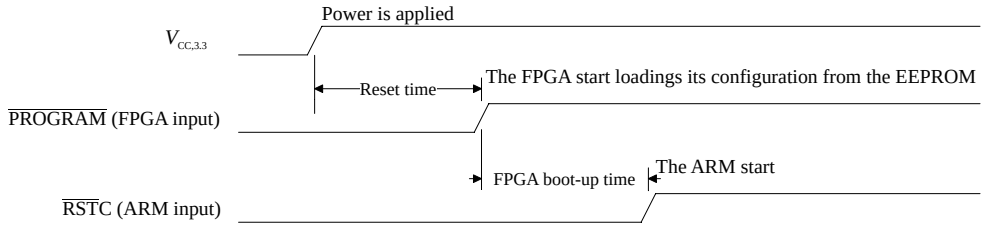


Figure 5.13: *The FPGA creates a power-up reset delay for the ARM. As shown, the FPGA does not deactivate reset before it is done loading its configuration.*

Configuration of I/O pins

The IOBs of the FPGA allow each output driver to be individually programmed for a range of low-voltage signaling standards. Besides, internal pull-up and pull-down resistors are available for each I/O pin. To minimize bus transients the appropriate drive strength level, i.e. output current, and slew rate control must be chosen for every outputs. High output currents and fast slew rates generally result in fastest I/O performances. However, it is recommended to use the slowest slew rate and lowest drive strength that meets the performance requirements for the application to avoid problems from transmission line effects.

In this configuration the FPGA uses the LVTTL standard designed for 3.3 V. Equation (5.1) in Section 5.3.2, page 61, is used to calculate the drive strength for individual outputs in proportion to the given load capacitance. The loads are not common for all outputs, thus different drive strengths are used. The configuration of each IOB can be found in the pinout report on the CD in the folder Source Code for FPGA.

For the $\overline{\text{WE}}$ output from the FPGA, the fast slew rate option is used since it is important that $\overline{\text{WE}}$ is negated before the address is being unstable during Read-out mode. Furthermore, this could be ensured by changing the drive strengths comparatively for $\overline{\text{WE}}$ and address outputs.

Each chip select input is provided with an internal pull-up resistor to disable all memory chips in the short period where the ARM is still reset after the FPGA has been configurated. Furthermore, if a chip select output from ARM looses connection to an input due to a bad connection the connections will not float. Hereby, the internal pull-up resistor ensures that a memory chip is not erroneously selected and thereby bus conflict can be avoided.

An internal pull-up resistor is also applied to the input connected to the PIO0 connection ensuring that the image sensor is in standby from the time the FPGA is configurated until the ARM has setup its PIO pins. Similar, on the input connected to the PIO1 connection an internal pull-down resistor is applied to ensure that the FPGA is in Pass-through mode just after the FPGA is configurated.

Protection Against Bus Conflict with the Image Sensor

Since the FPGA will enable the outputs of the image sensor if the PIO1 is set HIGH, it is critical that the ARM does not present data on the data bus during Read-out mode, where the PIO1 is asserted. To prevent bus conflict between the image sensor and the ARM it is possible to make the mode of the FPGA depend on $\overline{\text{CS}}$ inputs. Hereby, the FPGA can only be in Read-out mode if the PIO0 is asserted and the $\overline{\text{CS}}$ inputs are negated. However, this solution breaks with the one-hot state design and was first taken into account after the test of the FPGA was completed. A drawback from this is that test result can variate from the final design. It is though only expected this change will increase the mode changing delay.

It has been explained how a FPGA can emulate logic circuits, and how it is integrated into the microcomputer by dividing its functionality into a Pass-through and Read-out mode. It was specified how the Read-out sequence accomplish the DMA task, which time constraints were to be kept and consideration regarding the VHDL used for witting the configuration was outlined.

All hardware forming the microcomputer of the camera system are now established. Thereby, the microcomputer is prepared for software which will be designed in the next chapter.

6 Software Design

The purpose of this chapter is to present how realization of the requirements to the software running on the microcomputer is designed.

6.1 Process Overview

In this section, processes of the camera system are set up and their mutual relations are clarified. From the functional specifications in Section 3.2, page 42, the functionality of the camera system is divided into two superior tasks known as processes, which shall be handled. These processes are Communication and Image Handling. In the system design phase the functions of these tasks are described and the interfaces between them are explained. The modules within the processes are set up from the use cases described in Section 2.5.1, page 34.

On Figure 6.1 the control signals seen from the microcomputer is shown in the shape of arrows. The square boxes represents the external devices and the dashed lines frame the existing modules in the processes. The Image Handling process is activated when the camera system is powered up by

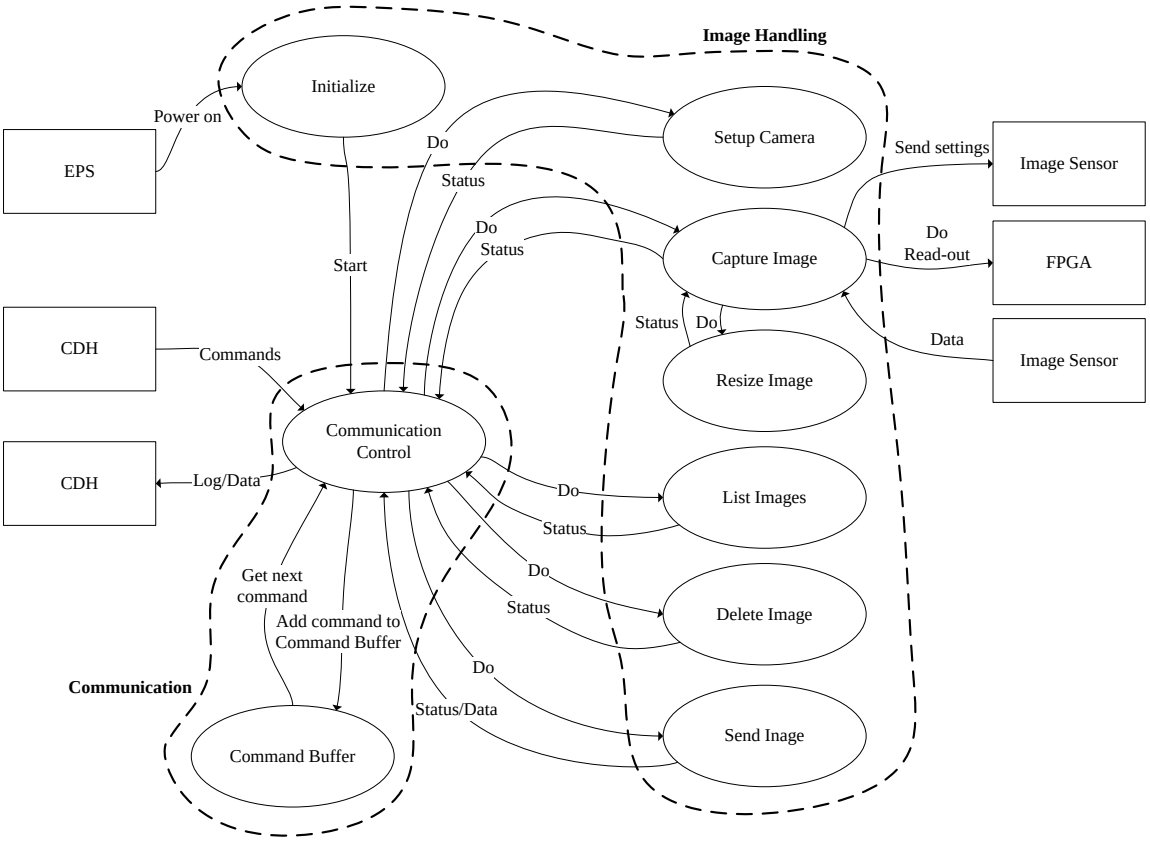


Figure 6.1: *The functionality of the camera systems software is divided into several modules. The processes Communication and Image Handling are superior tasks, both of them have related modules that enables them to solve tasks given to them. Within the module Capture Image in the Image Handling process a connection to hardware areas are defined. This is done to illustrate the origin of the data that appears in RAM in this function.*

EPS. Once initialized, the two processes should work in parallel. Only one module in a process can

be executed at a time. The initializing module in the Image Handling process sets up the software for communication; this is done to ensure that communication does not start before the system is ready to handle incoming messages.

The Communication process receives commands from CDH, which are addressed to the camera system and then saves these in the module Command Buffer. The Command Buffer must put commands in a queue, and is due to this implemented as a First In First Out buffer or simply FIFO. Simultaneously, the Image Handling process runs one command from Command Buffer if any is present. Modules in Image Handling can only be active one at a time; every module replies to Communication Control (ComC) when it ends. Communication Control is able to transmit data and status messages to CDH.

Notice the function Capture Image is connected with the external devices FPGA and image sensor so it can activate the FPGA that performs the actual readout of raw image data from the image sensor using DMA. Capture Image is also to transfer the settings to the internal registers of the image sensor. Thus, an image is always captured using the selected settings. Capture Image calls Resize Image each time a thumbnail is to be created.

6.2 Implementation of Processes

The conceptual Process Overview, Section 6.1, does not deal further with the implementation of the desired functions. Yet, the description of the design derived from the overview gives some important information about the tasks to be solved by the software.

The process overview indicates that the system must be capable of running the Communication process even when a requested command is executed by the Image Handling process. This is critical by the fact that the protocols of the HSN communication requires relative quick response time when a CAN packet is complete. During completion of a CAN packet this must be buffered for HSN to use. It is possible to achieve this by monitoring CAN activity with an interrupt line by using the interrupt handler in the ARM processor for protocol handling. Remember that the ARM is running at 40 MHz and therefore is capable of running several code lines from other processes during a bit detection on the CAN that on AAUSAT-II configuration only delivers bits in 125 kb/s. It is, however, considered by the project group an advantage to implement the software in an environment controlled by a scheduler. A scheduler is able to handle protection of registers and memory when shifting between threads. It allows the programmer to implement and test every functionality of a thread without having to worry about protecting the memory from other processes running. This will make the group able to develop the many functionalities of the software simultaneously in modules, which can be tested independently from each other.

Roughly a process with modules can be described as a thread with functions. However, a process can consist as an entity of more than one thread that cooperates as an object. During the DMA in the FPGA, the processor is unable to access RAM where the software is located, and therefore the scheduler must be turned off during this.

If controlled properly, the camera system should be able to run all the required processes simultaneously on the ARM, without loosing data or creating significant reduction of speed. On Figure 6.2 a visual interpretation of the software as scheduled threads are shown. It is noticeable that the command buffer is now reduced from being a functionality within the communications process to being a memory buffer (cmd_buffer), that the two threads can access at any arbitrary time. The buffer provides a uniform one way communications link between the two threads.

To ease the implementation of the threads, the names of them are shortened so that the name convention is "NAME"_thread. e.g. The MAIN_thread is the thread that provides the core functionalities, and ComC_thread is the Communications Control thread.

Software schedulers that allow parallel processes to be separately coded, run, and tested already exists in pre-developed embedded operating systems. Even though it is possible to design and implement a designated scheduler from scratch, it was found too time consuming by the project group. Instead an alternative solution was found. During the process of developing the AAUSAT-II OBC, similar considerations were to be taken, and here it was discovered that "eCos" was the best alternative found on the open source market [03gr731, 2003, p. 40].

Therefore, in this project the task of implementing eCos on the camera system became the basis for developing the software. To solve this task a stripped down version of eCos and a prototype

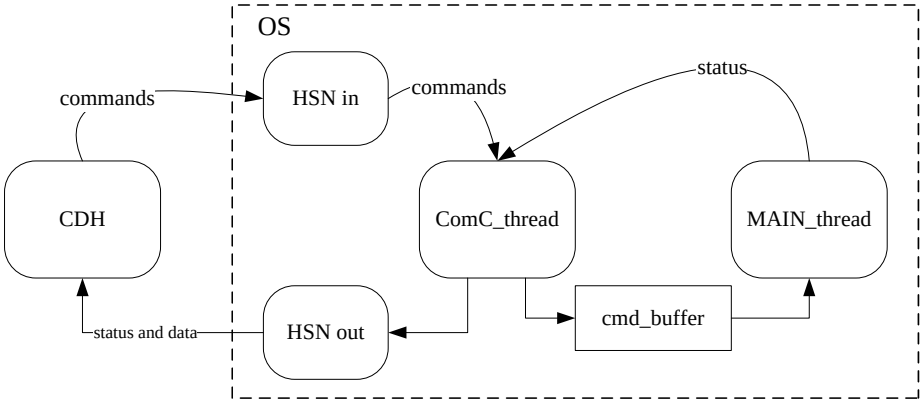


Figure 6.2: *The processes are here implemented as parallel running processes that can communicate along the arrows. The HSN threads are protocol handling threads.*

OBC were borrowed from the AAUSAT-II team, while the arrival of the prototype camera system PCB was awaited. This allowed for several tests of eCos and bootcode on a microcontroller similar to the one used for the camera system. A description of eCos and the functionality it provides are found in Appendix A.6, page 154.

An advantage gained by using eCos is that it supports all ANSI C functions; this includes malloc() and free() for dynamic memory handling. eCos is written in C++ and is predefined to run C code and be compiled with GCC [Massa, 2002], thus writing the application layer in C is an obvious choice. Though for some tasks such as writing the boot code it is considered by the group an advantage to use assembly.

It is chosen to use eCos for implementing the parallel processes as threads. In the following section it is described how the system is initialized to make it possible to run eCos on the hardware.

6.3 Initializing the System

The operating system provides a uniform base for executing the software. The system, however, needs to be setup for the specific hardware before it is ready for use; a low level boot code that initializes the ARM is implemented to prepare the hardware for running eCos and provides the possibility of booting different versions of the eCos based applications. This allows for eCos to be configured ad hoc, while developing the applications.

6.3.1 Microcontroller Boot Sequence

The purpose of this section is to explain what happens, when the ARM is reset, and which instructions are necessary to setup the ARM. This process involves setup of the external memory, choice of boot image, and starting up eCos.

When the ARM has been reset it is in reboot mode, meaning that it fetches its first instruction on $\overline{\text{CS}}0$ at address 0x0. From address 0x0 the exception vectors, for example IRQ, FIRQ, and data abort, are located. The address 0x0 is used for the Reset exception and contains an instruction which branches to the boot code. In the boot code the EBI and peripheral devices must be set up, followed by initializing eCos and setting the core clock. During setup, the ARM is set in remap mode, where the memory map is modified. The memory map in reboot and remap mode is shown in Figure 6.3, where it is shown, that only 1 MB external memory is accessible in reboot mode, and all internal RAM and peripheral devices can be used in both reboot and remap mode. The ARM has 4 kB internal SRAM even though it has 1 M addresses reserved to the internal SRAM in the memory map in both remap mode and reboot mode. This means that the 4 kB internal RAM is repeated 256 times.

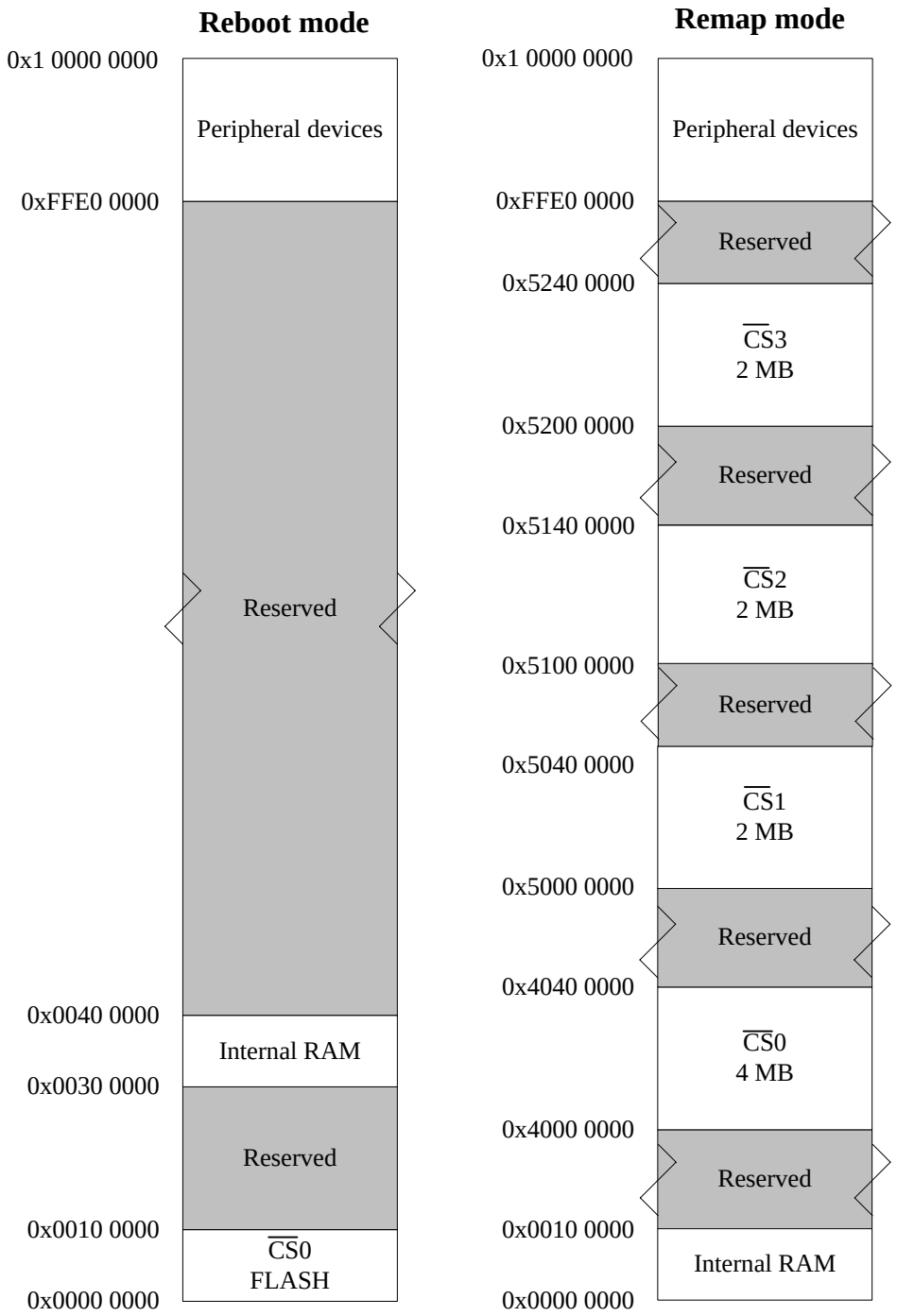


Figure 6.3: *Memory map in reboot and remap mode.*

Advanced Memory Controller

It is necessary to setup all external memory, which is done by loading base addresses, chip select enables, byte access types, data float output times, page sizes, wait state enables, number of wait states, and data bus widths into the Advanced Memory Controller Select Registers. The Advanced Memory Controller controls the physical layer of the bus interface called EBI. This is the first step of the boot sequence shown on Figure B.2 in Flowchart B.2, page 175.

The data bus width of the external memory devices must be selected. The ARM can either have a bus width of 8 or 16 bits, and it has a 16 bit data bus after reset. A 16 bit data bus is used on all

chip selects in this project, because the three RAM blocks and FLASH all have a 16 bit data bus.

Byte Access Type is set because a 16 bit data bus is selected. When Byte Access Type is set, the ARM accesses the external memory with one write signal, one read signal and two signals to select upper and/or lower memory bank.

The ARM can insert between one and eight wait states during the external access cycles. It is calculated in Appendix A.7, page 159, that five wait states are necessary to communicate with FLASH and two wait states are necessary to communicate with RAM. Due to lack of time it is chosen to use five wait states for both FLASH and RAM, hence this can be eliminated as a source of error. The ARM microcontroller has a maximum external address space of 16 MB; all used in this project, because a page size of 2 MB cannot be selected. This has to be addressed between address 0x4000 0000 and 0x7FFF FFFF in remap mode. It is chosen to set the base address of the FLASH at address 0x4000 0000 and to set the base address of the first RAM module at address 0x5000 0000 as shown in Figure 6.3.

The page size of each chip select also has to be set. It is chosen to set the memory page size to 4 MB on all chip selects, because the available page sizes are 1 MB, 4 MB, 16 MB, and 64 MB. The RAM uses only a size of 2 MB. This means, that the 2 MB is repeated one time [Atmel, 2005, p. 28]. The memory is therefore added to the memory map as pages. After the setup of the Advanced Memory Controller, the image which should be run has to be selected.

Choice of Boot Image

Different boot images can be uploaded to provide a possibility of software testing without erasing the default image. Thus a boot procedure is created that selects a bootable eCos image if possible.

The system uses a voting system to ensure that a power loss during software upload does not cause the system to crash at next boot-up. To be able to do this, a default image that is verified as bootable is loaded into a fixed memory address. Then a boot sequence is created that can select the new image, only when a series of validations have been passed. This is illustrated on the flowchart on Figure B.2 in Flowchart B.2, page 175.

After the advanced memory controller is set up, a jump code at a known location in FLASH is read. If this jump code matches a hard coded value, an index pointer to a row in the boot table is read; else the default image is loaded to RAM and executed.

When reading the index pointer the voting begins. The table and the index pointer is located in three blocks in FLASH that can be updated, and the voting ensures that it is still safe to read from the blocks so that only in the case when at least two out of three blocks with boot tables are similar, the value read is trusted. Every time something is read with voting, the possibility exists that the three tables are different and should that happen the default image is booted instead of the one currently being jumped to.

When the location is fetched, a check is done on the data found at the address to ensure that the image is an eCos image. When this is checked and validated, the size of the image is read from the table using voting. Again, any failure to validate that the image is an eCos image will force a boot of default software.

When these tests are passed the size is known, and all the code addressed by the table is loaded to RAM. During the copy of the image to RAM, a checksum is created on the copied data, and this checksum is compared with the one also found in the table. After the copy to RAM the actual booting sequence begins. The first part of the boot sequence is similar for both default and updated software; this is to remove the jump code from FLASH. Should any problem occur the procedure ensures that the default image is always loaded after a power reset. If it is wanted to boot the updated software instead, the jump code must be rewritten to FLASH before turning off power or resetting.

Initialization of the Peripheral Devices

When the desired image is copied to RAM the peripheral devices are setup. In this configuration seven Parallel In-/Output pins (PIO) are setup to control the mode of the FPGA and the state of the image sensor. To provide the communication between the FPGA and ARM, four PIO pins must be set as output pins and one PIO pin as input. The last two pins are used to communicate with the serial interface of the image sensor. These two pins are set as output pins, but during

communication with the image sensor one of these outputs are also used as an input to receive acknowledge and read settings from the image sensor.

eCos Boot

After the peripheral devices have been setup, eCos is booted which means that eCos is initialized and has to start the PLL and know which memory areas to use.

The PLL is setup to obtain a core clock frequency of 40 MHz from the 8 MHz external crystal. Therefore, the input frequency must be multiplied five times to provide the desired core clock, which is done by writing in the CM PLL Divider Register. An introduction to PLL is given in Appendix A.3, page 145.

eCos allocates memory in RAM and must therefore know where to put this memory. Therefore, the eCos memory map is set up to operate in address 0x5000 0000 to 0x5010 0000. This means that the remaining 5 MB RAM is reserved to image data needed when capturing an image and performing image handling.

The FLASH memory has to contain the software images, camera settings etc. To give an overview of the FLASH memory, the layout is explained in the next section.

6.4 Memory Usage in the Camera System

Most of the memory in use on the camera system is predefined by the project group; only a relative small area is in use for dynamic allocation. The dynamic allocation area, however, is controlled by eCos. The predefined areas and the structures used for describing them in the software is described in the following sections.

6.4.1 Layout of Memory in FLASH

The FLASH is by definition a place to hold data that should be persistent against power loss. The FLASH chip is build from blocks at a size of 64 kB, except from the last eight blocks that have sizes of 8 kB. When writing to FLASH an entire block is written from start to end by erasing it and then writing new data in it. When the satellite is in operation a risk exists of the subsystems temporary loosing power at any arbitrary time. If that happens during the writing or erasing of a block, the data within that block cannot be trusted when rebooting. Considering this, some protection against loss of functionality after a temporary power loss is to be implemented.

On Figure 6.4 the layout of fixed memory in FLASH is illustrated. Some information is more critical to the software than others. The compiled software image itself is for instance a very typical example of something that must not be corrupted of a power loss. The software running on the camera should, however, still be easy to update especially during debugging. A system is created that is prepared to handle software uploads in space. However, it is not fully implemented in this prototype of the camera, as new software so far only is uploaded from the debug interface. This is to be further developed to function on the HSN protocol, but should not be done before the OBC for AAUSAT-III is designed.

The default settings for the camera system is never to be overwritten and it is decided to have set of default settings for the image sensor just as there was a default software image in FLASH that must never be altered.

The image info in the image headers are responding to the image data slots. The state of this data and image info is indicated in FLAGS. On Figure 6.4 the relation between the FLAGS and the slots are described as colored entities. A flag in FLAGS is set if the data in an image slot is to be interpreted as a present image. A cleared flag indicates that no data is present.

Both the boot table and the image info in the image headers is data that the system needs to update during some operations and both contain data which should be protected from power loss. Even though they are to be updated a complete loss of this info could cause major problems. These data types are protected from complete erasure using a triple block replica of the data. To explain how this protects the data an example is shown using the image header data.

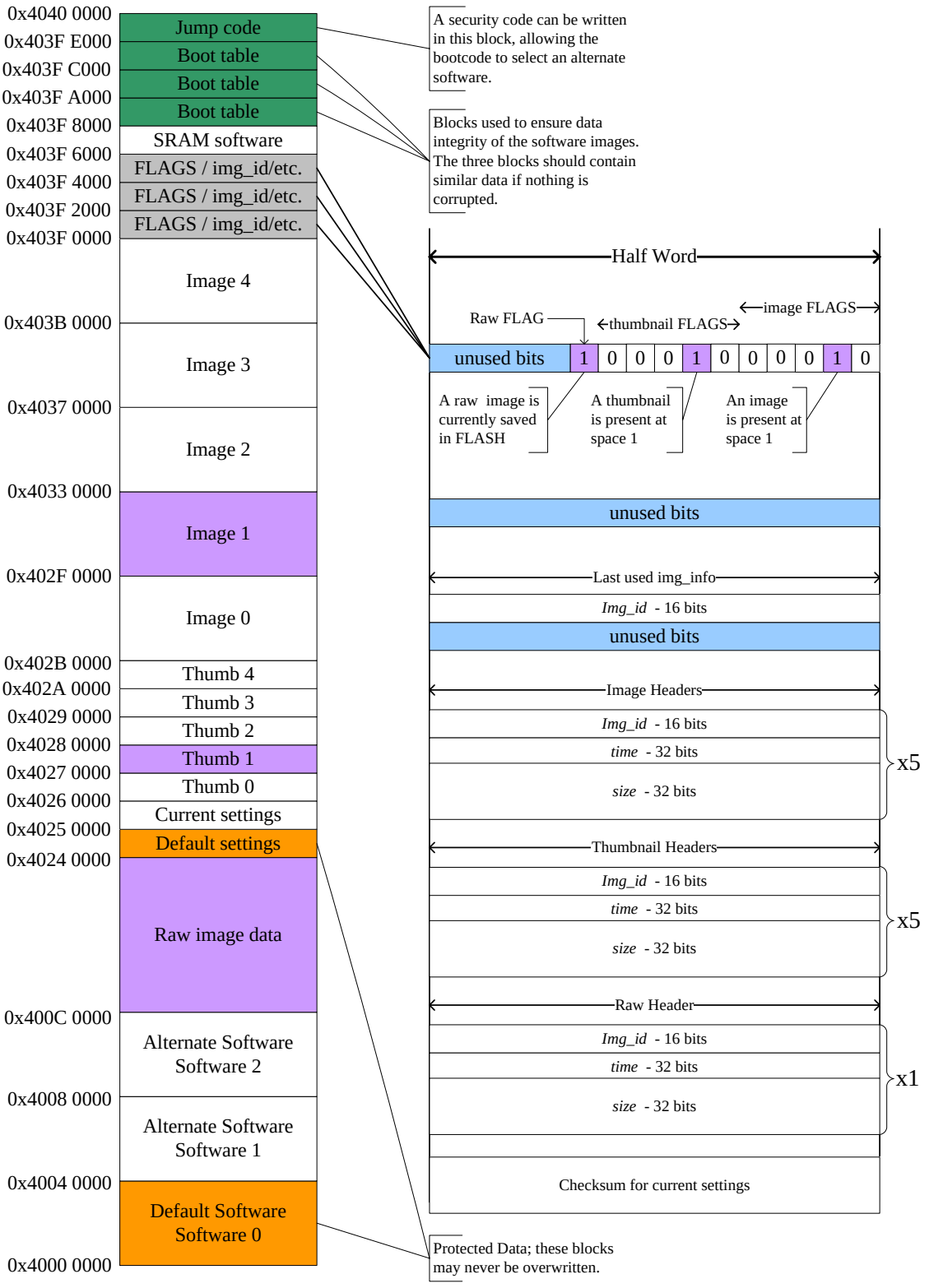


Figure 6.4: *The layout of FLASH indicates where different types of data are stored. Color codes should be read as follows: Orange is permanent data never to be erased. Dark green is data relating to bootcode used for verifying software versions. Gray is image info; the software uses this to header info to quickly determine the state of the image data in memory. Lavender is image data, the content of these blocks are described in the image headers.*

Three FLASH blocks are used for FLAGS along with the last used unique image identification counter (*img_id*). The counter is incremented just before an image is taken and is during capture stored in the image info in the appropriate image header. After a reboot, the image identifier to be used from now on is always set to start from the value found in last used *img_id*. Should power be cut during the saving of an image, the last used image identifier ensures that the image identifier is not used twice.

A procedure is developed to ensure that the system will be able to know which of the images stored in FLASH that are valid. At a worst case scenario the image latest produced could be lost, but this is preferred instead of risking transferring corrupt image data.

The procedure when saving an image is:

- The first of the three FLAG blocks is erased.
- The last used *img_id* is incremented and saved in the first block and the FLAG, indicating if this image exists, is cleared.
- The middle of the three FLAG blocks is erased.
- The incremented last used *img_id* is also saved in the middle block and the FLAG, indicating if this image exists, is cleared.
- The third of the three FLAG blocks is erased.
- The incremented last used *img_id* is also saved in the third block and the FLAG, indicating if this image exists, is cleared.
- The image data is saved in the designated image blocks.
- The first of the three FLAG blocks is erased again.
- The erased block is rewritten with the FLAG for the image set, and the image header.
- The middle of the three FLAG blocks is erased again.
- The erased block is rewritten with the FLAG for the image set, and the image header.
- The last of the three FLAG blocks is erased again.
- The erased block is rewritten with the FLAG for the image set, and the image header.

When accessing the FLAGS, the three blocks are used to make a qualified guess on what block of FLAGS and last used *img_id* is containing valid data.

First a check is made to see if the three blocks are similar if so they are assumed valid. Should only one of the three differ from the rests, it means that either power was cut during the saving in the first or last of the three blocks or that a write error and or bit flip has occurred. No matter which is the case, the two blocks that are similar are then interpreted as valid and used to overwrite the one that is differing.

If the power was lost during the saving of data in the first block, this method will indicate a cleared FLAG for the image, and access to the last image stored is lost, but the system will except for that function properly. When the next image is captured it will take the place of the image lost.

Should, however, power be lost at saving the middle block of the three, a problem occurs at reboot since all three blocks would contain different data. If this happens the last used *img_id* is fetched from the first and third block, if these are enumerated properly so that the one in the first block is exactly 1 higher than the one in the third, it should contain valid data. This is then used to overwrite the others, ensuring that the image last taken is not lost to the user. If they are not enumerated properly the third one is used to overwrite the second and the first one. Verification of Boot tables are using a similar method.

6.4.2 Image Info Structure in FLASH

In this section some of the data types used in FLASH for image info are described. None of these are critical for the functionality of the software, meaning that if they are lost due to a power loss the software will still function even though data is lost.

On Figure 6.4 a description of the data fields used for saving an image is found. The image can optionally be stored as both raw image and thumbnail, but is by default always saved as a compressed image. The camera system can hold up to five compressed images and up to five resized compressed thumbnails. An image identifier *img_id* at 16 bit is stored in a header that is representing each of the images stored. This is used to allow the user to identify the image knowing the location of the image. Other fields that are embedded in the header is *time* and *size*. The *time* field contains a time code from OBC telling when this image was requested. The *size* is a 32 bit integer containing the size of the image data. This is put there for handling images that due to compression takes up different amounts of space in FLASH.

The thumbnails are given an entire 64 kB block for each, even though this is somewhat a waste of memory. This ensures that no other information can be damaged except from the thumbnail itself in case of a power loss during image saving. The size for a compressed image is too large to keep in a single block, remember that the demands to the system was maximum 256 kB **§FUNC5**. Therefore, 4 blocks of 64 kB is given to each image.

The *img_id* can be used to request the image from the camera system without ever knowing the location in FLASH. This decision was based on the assumption that the designated user on MCC or CDH have no advantages to gain from direct access to the FLASH. Instead using an unique identifier that is similar on both the thumbnail and the corresponding compressed image, ensures that the user easily can locate a subject. It is done to allow the user to fetch a thumbnail and get an idea of the subject, and then afterward using the same identifier, get the high resolution image if the subject is good. There is also allocated room for a single image in the raw format to allow for a better compression method run later on MCC.

6.4.3 Image Info Structure in RAM

The image info headers found in FLASH contain information about the image data in FLASH. These are located in designated words and in a known structure as was illustrated on Figure 6.4. To handle this when the user requests information about images or perhaps image data, a structure is created that enables the software to handle the image easily. Instead of the known datatypes in ANSI C, the U(SIZE) datatype is used. They represent unsigned (SIZE) bit types; i.e. U16 is an unsigned 16 bit datatype.

The structure *img* contains all the necessary information about an image. It is used both to generate a list of image info, and for containing image chunks if an image is to be send. It is build temporarily when needed of the following datatypes:

- U16 chunknum
  Is the variable that is used to define which chunk number is referred to in chunkdata pointer pf the structure. A compressed image can be larger than the chunks of 10 kB defined in **§FUNC6**. This makes it important to identify the specific chunk being handled. The chunknumber is found by counting chunks of data of 10 kB from the start address till there is no data left. When a list is requested this variable is set to 0 indicating the first chunk.

- U16 img_id
  Is the variable used to identify which image is pointed at by the chunkdata pointer. It is the *img_id* found in FLASH in the header. This is used both in generation of the list and in the image chunk sent.

- U32 time
  Is a variable containing the time of capture information in the image header in FLASH. This like the *img_id* is send both for a chunk and for the list. The time stamp is stored in UNIX formatting, this is also known as POSIX which eCos supports.

- U32 size
  Is the variable the camera system uses to know exactly how much space the image data takes up. It is used to calculate the number of chunks and is also sent in the list.
- U8 type
  Is the variable the camera system uses to handle the three formats that can exist for an img_id. It is send in the list to inform the user which *img_id* is saved as a raw and, how many of the compressed images have a thumbnail stored.
- U8 *chunkdata
  Is the actual pointer to the data. It is pointing at NULL if a list is generated. When this points to NULL the chunknum is not sent either.
- struct img *next
  This pointer can be set to NULL or to point at a structure similar to this one. This is used in list generation to inform the transmitting process where the next element in the list is located. Should a chunk with data be sent with this pointer set to a structure, the structure will be send afterward, as was the case with the list. This allows for the user to request an entire image.

6.4.4 Software in RAM

At boot time, the compiled software including eCos is copied to RAM before execution. The image of the software currently running could therefore be updated without changing software first. Another advantage is that RAM should be considerably faster than FLASH.

When an image is transferred to RAM from the image sensor, the hardware does not allow for the software to access either RAM or FLASH due to the DMA implementation. The eCos scheduler is therefore needed to be halted so that the interrupt service routine does not try to fetch code. The FPGA still requires some control from the ARM during image capture. This requires some form of software to be running on the ARM even when the scheduler is halted, the applications in eCos are like the scheduler in need of having access to its code in RAM. The chosen ARM here presents the solution, since the internal 4 kB SRAM presents an opportunity of storing running a limited amount of code. Two options are obvious, either the specific thread running in eCos for image capturing should be run from SRAM allowing it to continue when the scheduler and the RAM is disconnected, or a switch to an alternate software should be done.

For flexibility of the system, it is decided to compile a smaller alternate software that can be loaded into SRAM from the running eCos. This allows for this specific function to be updated without having to reboot the camera system. The alternate software is due to its behavior in relation to eCos titled “SRAM Software”. The SRAM software is described separately since it does not affect the flow of the multi threaded software. When switched to SRAM software, the eCos scheduler is no longer activated, and this ensures that no attempts are done to access external memory, if not programmed in the alternate software.

The persistent memory is divided into different types, some of it, like the default settings and default software is permanent, others like the boot tables are protected by voting and therefore are found multiple places in FLASH. Temporary data is allocated in RAM. The handling of the memory is done from the threads mentioned earlier, and they are treated as objects that can interconnect with each other. The flow of the two major processes is described in the next section.

6.5 Software Flow of the Multi Threaded Applications

The software platform to be used for implementation has been prepared to multi threaded applications and the basic memory layout have been described. Still, the application software handling the functionality of the use cases needs to be implemented on the system. To get an overview of how to implement the functionality in the threads and connect their functions to a completed software, a set of flowcharts are created in the form of UML active state charts [Lee and Tepfenhart, 2001]. This is also known as activity diagrams [OMG, 2004, 402]. The designed flow of the software is described in the following sections.

The flowcharts is found in Chapter B starting from page 173. The charts are located on A3 formatting and can be folded out, so that the resulting chart can be seen while reading the documentation of the design. Each flowchart is placed on an individual page. Section B.1 describes how to read the symbols used in the flowcharts.

6.5.1 Thread Design

The threads run as parallel processes and communication between them is to be considered as signals from other components. The functions in each thread are sequential for each thread and a software flow can be derived for each thread.

A flowchart is a powerful tool to get an idea of the implementation of the functionalities required for a thread. The more specific logic decisions depending on the functionalities are here defined so that the threads behavior can be predetermined before the code is implemented. The elements of the software which are important for the illustrated flow are described as text or pseudo code. The syntax used corresponds to ANSI C due to the fact that this is the chosen language for the implementation.

Notice that the name conventions are small caps for arguments and functions, and large caps for objects such as threads or functions that include several sub functions. There is a few exceptions like ComC_thread, these are contractions of multiple words, in this case Communication Control.

MAIN_thread

The flowchart describing MAIN_thread is found on Figure B.3 on page 177.

The MAIN_thread is started as a part of the eCos initialization, as described in Process Implementation in Section 6.2 and shown on Figure 6.2, page 77. The thread is intended to fetch new commands in a command buffer (cmd_buffer) and execute them in the order they where stored. The command buffer is filled by the communication thread (ComC_thread). Before the MAIN_thread can check the buffer the ComC_thread must naturally be spawned.

As the ComC_thread is initializing, MAIN_thread cannot get any commands from the command buffer; not only because no commands are put there by ComC_thread yet, but also because the MAIN_thread simply does not know where the buffer is being allocated, since this is done dynamically. The MAIN_thread is set to wait for the communications channel to be ready, and afterwards move along, and upload the SRAM software so that it will be ready when later needed for image capturing.

When ComC_thread has created the *cmd_buffer*, the MAIN_thread can continue and enters the main loop in which it is to remain until the system is shut down. The loop ensures that the next command in the buffer is read after the previous command was carried out. The loop starts of with a semaphore holding the thread waiting until a command is present in the buffer. A limited amount of commands are accepted by the system. If an unknown command is received an error message must be returned to inform the CDH that something is wrong. The validation is done from MAIN_thread, since it is in this thread commands are executed.

If a received command is valid, the command is executed. A status message is returned to inform the user if the command was executed successful, this status is expected to be put in the CDH log that can be retrieved from MCC on Earth. The flow of the commands are described as separate functions in Section 6.5.2, page 86. A list of commands can be found in Appendix A.10, page 163.

The thread should never run the same command twice, so the commands are removed from the cmd_buffer every time the loop is returning to the beginning.

ComC_thread

The flowchart describing ComC_thread is found on Figure B.4 on page 179.

The ComC_thread is initiated by the MAIN_thread. The purpose of this thread is to ensure that commands from the user are put in the *cmd_buffer* for later execution. Also status messages from the MAIN_thread is to be returned through this thread. The thread exists to ensure that communication is possible no matter what the MAIN_thread is working on. During the shift to the alternate SRAM software in capture image, communication is though not available. This is allowed

due to the timeout configuration possibilities in HSN. Controlling the communication also includes the handling of data transmission. The ComC_thread must therefore be able to transmit data when a list of images, or image data is required by the user.

To handle the HSN protocol and the CAN protocol, extra threads are spawned from the ComC_thread. The thread is therefore designed to handle command and data flow between the threads. Both the MAIN_thread and the HSN protocol threads puts information in *ComC_mailbox* and from here the ComC_thread analyzes the message and redirects the command or data to the appropriate recipient.

Before entering the main loop of the ComC_thread, the address for the *cmd_buffer* is send to the MAIN_thread. When entered the main loop the thread awaits that something is put in the mailbox.

ComC_thread is to handle very different kind of messages, so the messages must be formatted so that a message type is explicitly defined.

The message types are:

- STATUS
  The status that MAIN_thread reports back from the functions. This should simply be relayed with arguments attached.
- POINTER
  Pointer to an img structure; this datatype is described in Section 6.4.3, page 83. The data contained in the structure is to be sent on HSN. There exist two possibilities of data to send, either it is image data, or it is a list of image headers. If the *chunkdata pointer is a NULL pointer, the img structure is a part of an image info list, and the chunknumber is not sent with the information. The information obtained from the header and the type is, however, sent. Should there be pointed at something else it is assumed to be data, so the data pointed at and the chunknumber is send. As long as there is pointed at a structure img with the *next pointer, the element pointed at is to be send afterward.
- COMMAND
  Commands are a type that the HSN threads will set if anything is picked up on the network for the camera system. It should be put in the *cmd_buffer* to be available for MAIN_thread that will handle the validation and execution of the command.
- PING
  PING is a message type that the HSN thread will reply with a PONG to CDH.
- HOUSEKEEPING POINTER
  Pointer to a string containing housekeeping data; this string contains the temperature by the image sensor and all *img_id*s.

6.5.2 Function Design

For each module in a process described in Section 6.1 a corresponding function in the appropriate thread exists. Some of the functionality required by the overall design is already implemented in the thread design, the rest of the sequential functions in the MAIN_thread are so complex that they have been described on separate flowcharts. This section describes the flowcharts of those functions.

int cam_setup(void *data, U8 length);

The flowchart describing cam_setup is found on Figure B.5 on page 181.

The camera system is to avoid unnecessary power consumption, so the image sensor is only powered up when an image is to be captured. The settings of the image sensor used to capture the image can therefore only be transferred immediately before an image is to be captured. However, to allow for a flexible communication with the CDH, settings to be used for the next image capture should be allowed to be transferred to the camera system at any arbitrary time.

The camera system is designed to allow for a user to transfer commands at all times; this also include the settings. As a consequence a function is needed to handle the incoming settings and store them in FLASH. The settings transferred are specified in the user manual, in Appendix A.10, page 163, to be sent in a continuous bit stream, containing a formatting as can be seen on Figure 6.5.

When transmitting settings for the image sensor register 0x02, register 0x23 must also be transmitted as register 0x02 is dependent on register 0x23 so they are validated against each other before they are saved in FLASH. The image sensor has 34 registers that can be manipulated by a user, so these are all made available for the user of the camera system to handle.

The reason for the validation of the settings is that the image sensor does not report an error if a setting is out of bounds. Instead it saves the valid bits and ignores the invalid bits; the user could therefore be set in a situation where he does not know what a setting is set as, if the camera system does not stop him.

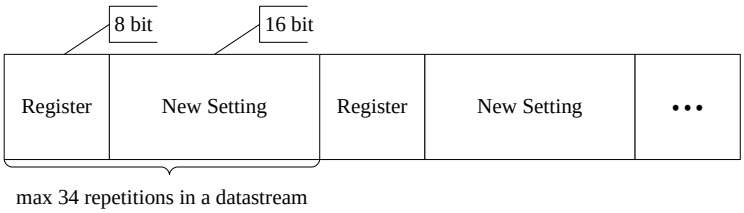


Figure 6.5: *The altered settings can be sent in a stream of settings. All 34 changeable registers can be contained in the stream in random order, as long as no register appears twice.*

The settings transferred are saved in FLASH and a checksum is made to handle possible errors in storing. The checksum is also stored in the current settings block, this allows the system when handling current settings to easily check if they are valid. The reason for using FLASH to store the settings and checking them afterward is that settings successfully transferred are verified as the ones in FLASH and when this is verified, the settings are never forgotten by the camera system even if a power off should occur.

The location of the current settings is illustrated on Figure 6.4, page 81. A set of default settings are also kept in FLASH, so that a simple command can return the system to a known configuration, if this should be desired by the user.

int cam_defaults();

The flowchart describing cam_defaults is found on Figure B.6 on page 183.

The function cam_defaults is implemented to reduce the necessary communication on the satellite uplink. Of course a return to default can be achieved using the cam_setup() and transmitting all the default settings, but this would require a stream of settings containing 104 bytes. Default settings are described and shown in Appendix A.9, page 162.

A return to default settings is, however, much easier to accomplish by transmitting a single command. The design generally avoids the use of magic numbers and in this case the functionality is evolved from the memory layout in FLASH. Here the current settings like all other data can be corrupted if the power is lost while overwriting them. To avoid this possible error to prevent the camera system from being able to capture images, a set of default settings is at all times stored in the FLASH. Checksums for the current settings are also stored when the settings are changed, which allows the system to fall back to default settings, if a write was not completed. Having default settings saved in FLASH opens for the user to fall back to defaults by simply overwriting current

settings with default settings. To allow this, a function is implemented to handle this overwrite, so that it can be called directly as a HSN command.

cam_defaults generates checksums to check if the overwrite was a success and generates an error if it was not successful.

int cam_capture(int time, bol thumb, bol raw)

The flowchart describing cam_capture is illustrated as a flowchart that is found on Figure B.7 on page 185.

When the user sends the command to capture an image, the function cam_capture is called in MAIN_thread. It takes three arguments: *time*, that is the time when the OBC relayed the command, *thumb*, which is a logic true or false argument that indicates if a thumbnail should be generated also, and *raw* that have a functionality similar to the *thumb* except that this one results in saving the raw data before compression.

For each time an image capture is performed, the last used *img_id* in FLASH is incremented. This is done by this function before the image sensor is initialized and settings are transferred to ensure that any capture gets a unique identifier.

A check of the temperature of the image sensor is also done before initializing it, this is done using the serial connection to a digital temperature sensor found on the board from Devitech ApS. Should the image sensor not be within the allowed temperature range the function never initializes the image sensor but instead returns an error. The temperature range can be set in a header file before compiling the software, and by default it is set to 0 to 60 °C, as this is the range specified in the datasheet for the image sensor.

For each time an image capture is done a compressed image is produced. However, it is to be produced as the last of the image formats produced, if more are requested. This requirement follows of the hardware design where the amount of memory does not allow to keep the raw data while compressing it to the compressed image. For each image formatting different procedures are run.

First, the *raw* argument is tested. Should this be requested the old raw data in FLASH is marked as invalid by resetting the flag in the FLAGS field. Then the data is replaced by the new raw data before the flag is set again. The validation of the raw data in FLASH is easily done using the raw data still found in RAM. Only if the data saved is valid the flag indicating a valid set of raw data is set.

Secondly, the thumbnail argument is tested. Should a thumbnail be requested a resized copy of the raw data in RAM is created. This is possible to do without damaging the original raw data, since the raw has not been interpolated yet. The resize method uses an average of pixels in an array to create the pixels in the resized thumbnail. As so, no interpolation is needed for the resized images.

The compression algorithm is run on the resized data and while compressing it saves the data in FLASH. When the thumbnail is compressed, the flag of the thumbnail in the FLAGS field in FLASH is set. The compression algorithm also generates a checksum during compression. This is checked before the flag is set, making sure that corrupted data is unlikely to be validated by accident.

Thirdly, the image is interpolated and afterward compressed; again a checksum is performed to validate the compressed image before setting the flag in the FLAGS field.

For each requested image format that is saved, a bit is set or reset in a status indicator. This status indicator is used to combine an error message, if *raw* or *thumb* is not requested, it will result in a set flag for the format as no errors occurred.

Before fetching data and right after the initializing of the image sensor, the entire operating system is halted and another software (SRAM software) takes over the control of the microcontroller.

The software returns the program counter to the exact point where the software was left as soon as the raw data is transferred by the FPGA from the image sensor to RAM.

During initialize_image_sensor the settings for the capture is transferred using the two PIO pins that are connected to the image sensor, this was also used for the temperature sensor; more information about this protocol implementation is found in Section 7.2.1, page 92. More information about interpolation and compression is found in Section 7.5, page 94, and Section 7.7, page 98.

int cam_list_images()

The flowchart describing cam_list_images is found on Figure B.8 on page 187.

The concept of the cam_list_images is to provide an easy to use overview of the images stored in FLASH. The data stream this function builds is a linked list of *img* structures. The data pointer in the structure is though set to NULL, this will make sure that only image info is sent, and not a part of the actual image data. It is as so only in cam_list_images that this pointer is set to NULL, whereas it in e.g. cam_send_image is set to point at data.

The structure contains the *img_id*, the *time*, the *size*, the *type*, and a pointer to the next element **next*. Since the size of the structure and order of this are well defined, it is easy to decompile the data stream into structures of the same form as they where sent.

When ComC_thread handles the **next* pointer it is of course replaced with the data contained in the structure pointed at. So this must be considered when decompiling the bit stream. As a result, a list is generated that contains all the necessary information about the images that is needed to identify any image in the list.

The *img_id* is similar on images, thumbs and the raw that is captured at the same time, and initially a thumbnail can be downloaded to get a feel of the subject before the much larger image format is fetched. This will save time and bandwidth for the user that does not have to waste time transferring the entire image if the subject chosen is unfortunate.

int cam_delete_image(int img_id)

The flowchart describing cam_delete_image is found on Figure B.9 on page 189.

When an image has been downloaded to MCC, the use cases specified that it should be possible to delete the images; refer to Section 2.5.1, page 36. This is made easier for the user by the use of the *img_id* and *time*. When the information is used to identify the image, the *img_id* can be used to select it and delete it afterward.

When requesting the deletion of an *img_id* the image and the thumb with the matching *img_id* is deleted. However, an image stored in FLASH as raw cannot be deleted. This decision was taken due to the fact that only one raw image can be stored at a time, and that cam_capture should always be allowed to overwrite this. So there is no reason to delete it when it is not protected from being overwritten at next capture.

int cam_send_img(int img_id, int chunknum, U8 type)

The flowchart describing cam_send_img is found on Figure B.10 on page 191.

When requesting an image it is necessary to specify the *img_id* of the image to identify it. Furthermore, the function should be able to send thumbnails, images, or even the raw data stored in FLASH of the *img_id*. To distinguish between them, the argument must also be sent when requesting an image transfer. The types are:

0 Compressed image.

1 Thumbnail.

2 Raw image.

Any of these "type" arguments will return the entire image in the format wished for. It is send in numbered chunks of maximum 10 kB, since this is required in **§FUNC6**. A header with information about the chunk is send with the data creating a packet of 10 kB plus the header. The header is the image info found in the img struct and a chunk number.

cam_send_img calculates the total number of chunks from the size identifier found in the image header in FLASH. The last chunk in an image can be smaller than 10 kB. So the size sent in the header is the size of the chunk; not the size of the image. For each chunk sent the chunknumber in the header is set to the number of the chunk.

The function uses this to generate a linked list with structures similar to the ones created by cam_list_images, but with the data pointer actually pointing at image data. Should an error later occurs and a chunk is lost in the transfer, it can be retrieved by using the type argument to specify that a chunk number is requested. If this is the case the chunknum integer must be filled with the chunk number requested. The types returning a single chunk are:

3 Image chunk.

5 Raw chunk.

The chunk numbering is set like this as it gives advantages in implementation. As you simply can subtract 3 to get the type of image, the chunk is found in.

The header information found in the list contains the total size of the image. This can be compared to the sum of the sizes of the chunks, should any doubts later be raised about the existence of the last chunk, if this is very close to 0 kB.

int cam_housekeeping(U32 *hkpointer)

A flowchart describing cam_housekeeping is found on Figure B.11 on page 193.

The functionality of this implemented module is to provide the user and CDH with valuable data about the condition of the camera system.

Even though this is not found in the use cases in Section 2.5.1, it is a functionality used by all the subsystems on AAUSAT-II. Thus, it is expected that AAUSAT-III will have a similar need. The housekeeping data is on AAUSAT-II transferred by CDH in each beacon. This allows any radio on Earth to collect the data and send them to AAU for decoding and analysis.

In the prototype it collects temperature data measured at the image sensor and the FLAGS word from FLASH. This data is intended to be put in a beacon and in the CDH log, but can also be called directly by the user. More information can easily be added to the data stream if this should be needed in a later edition.

The flow of the threads and the functions of the threads are described to provide an overview of the software. The subfunctionalities are still not described as they are not important to the flow of the software, but still they can present an enormous task to implement. In the following chapter it is described how these subfunctions are implemented.

7 Implementation of Subfunctions

The purpose of this chapter is to document the implementation of the subfunctions included in the the flowchart of cam_capture(int time, bol thumb, bol raw) shown in Chapter B on Figure B.7, page 185. When it is necessary, suggested solutions are examined, a solution is chosen and the implementation of this is described.

7.1 Read-out Temperature

Before initializing the image sensor the temperature of the image sensor should be measured to make sure that the temperature of the image sensor is within its operating temperature range. This is handle by read_temp(*temperature). The task is done by following these items:

1. **ARM enables the temperature sensor.**
   ARM addresses the sensor and flips the shutdown bit pulling it out of shutdown mode.
2. **ARM reads the temperature by the image sensor.**
   The ARM must read the temperature by reading register 0 of the temperature sensor.
3. **ARM sets the temperature sensor in shutdown mode.**
   The ARM must set the temperature sensor in shutdown mode after reading the temperature. This is done to reduce the supply current to less than 1 μA [Instruments, 2005, p. 6].

The serial protocol used to communicate with the temperature sensor is equal to the protocol used to communicate with the image sensor, and is therefore described in Section 7.2.1 [Instruments, 2005].

7.2 Initializing Image Sensor

The image sensor must be initialized before it can capture an image and transfer the image data to RAM. The initialization is performed by initialize_image_sensor() and includes powering up the image sensor and transferring the desired settings to the image sensor. The following must take place:

1. **ARM enables a 10 MHz clock-output on T0TIOA0.**
   The ARM must enable its T0TIOA0 output pin and set a 10 MHz clock onto it. The pin is connected to the image sensor through IS-CLK and is needed to operate the image sensor. The clock must be enabled before the serial communication between the ARM and the image sensor can happen. The ARM must pull the image sensor out of standby before it can start serial communication. Notice that the PIO0 is passed through the FPGA to IS-STB.
2. **ARM activates reset mode in the register of the image sensor.**
   The image sensor is put in reset mode by setting register 0x0D bit 0. This implies that all register settings in the image sensor are set to default settings of the image sensor and Chip Enable is automatically asserted [Micron, 2005, p. 16].
3. **ARM deactivates reset mode in the register of the image sensor.**
   The image sensor is pulled out of reset mode by clearing register 0x0D bit 0 and thereby making it possible to write new settings.
4. **ARM transfers current settings from FLASH to the registers of the image sensor.**
   Current settings located in FLASH are transferred through the serial interface to their corresponding registers of the image sensor.

To make the serial communication between the ARM and the image sensor it is necessary to know how the serial protocol works. Therefore, the serial protocol and how the serial communication is programmed is explained in the next section using the data from the datasheet of the image sensor.

7.2.1 Communication Between ARM and Image Sensor

The communication between the image sensor and the ARM is accomplished by using a two-wire serial communication protocol dictated by the image sensor. The serial communication is used to write settings to the image sensor and to reset the image sensor. When the ARM writes one setting to the image sensor, the timing follows the timing diagram shown in Figure 7.1.

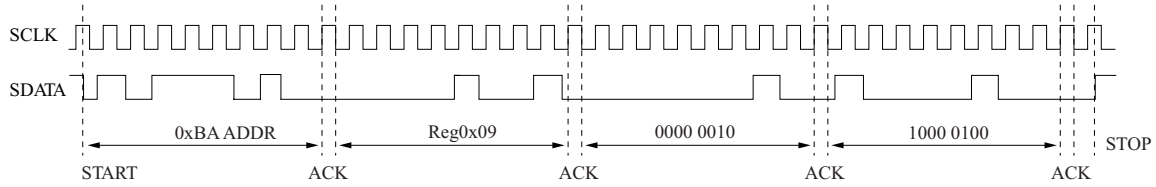


Figure 7.1: *Timing diagram for a typical write cycle [Micron, 2005, p. 34].*

In the timing diagram a start bit is written followed by eight bits determining whether the image sensor is in read or write mode. Seven out of the eight bits are device address bits and the eighth bit determines the mode; either read or write mode. These eight bits are followed by an acknowledge bit sent by the image sensor. Now eight bits, determining what register to write in, are sent from the ARM followed by an acknowledge bit from the image sensor. Subsequently, the ARM has to send a 16 bit setting to the image sensor, separated and ended by an acknowledge bit sent from the image sensor. Once no more settings are to be transferred, a stop bit is sent to the image sensor.

The entire communication has to keep up with the time constraints shown in Figure 7.2 according to the datasheet of the image sensor. Figure 7.2 shows that SDATA has to be stable when SCLK is HIGH during data transfer and that SDATA has to change when SCLK is HIGH to generate a start or stop bit.

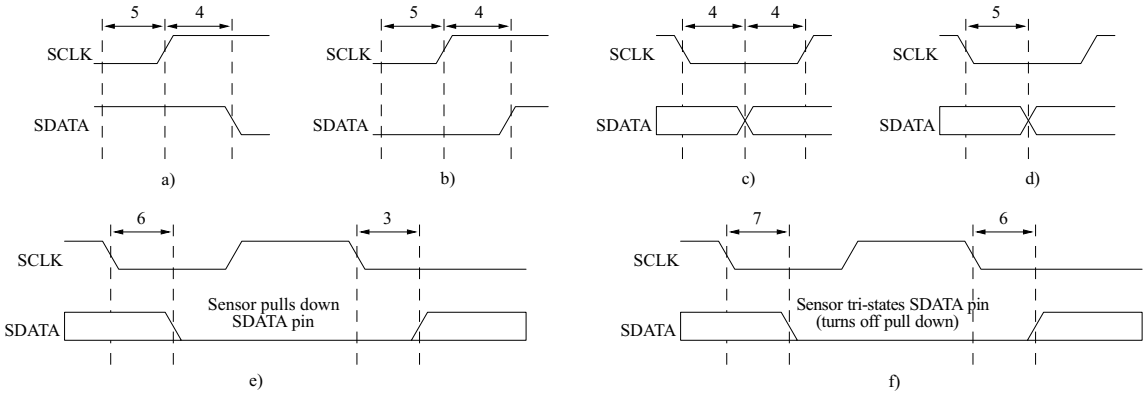


Figure 7.2: *Timing of the serial interface between the image sensor and the ARM. The time is specified in minimum image sensor master clock cycles. a) Start condition timing, b) Stop condition timing, c) Data timing for write, d) Data timing for read, e) Acknowledge signal timing after write, f) Acknowledge signal timing after read. [Micron, 2005, p. 37]*

When the ARM writes settings to multiple registers the communication protocol allows some short cuts. These short cuts are used when current settings are written to the image sensor. Once a setting has been written to a register, the register address automatically increments so the following 16 bits are written to the next register. Furthermore, only one stop bit, one start bit, and one byte, determining the mode of the image sensor, is needed to write into multiple registers, because the write mode is stopped by writing a start or stop bit.

To test whether or not the serial communication works, a function which can read the settings of the image sensors registers has been composed. This function follows the timing diagram shown in Figure 7.3 and keeps the time constraints shown in Figure 7.2.

A read sequence starts as a write sequence by writing a start bit followed by 8 bits of register address. After the 8 bits of register address the ARM has to send a start bit and set the image sensor in read mode. Once this has been done the flow of the read and the write cycle is identical, besides the fact that the image sensor now sends the settings and the ARM sends the acknowledge bits.

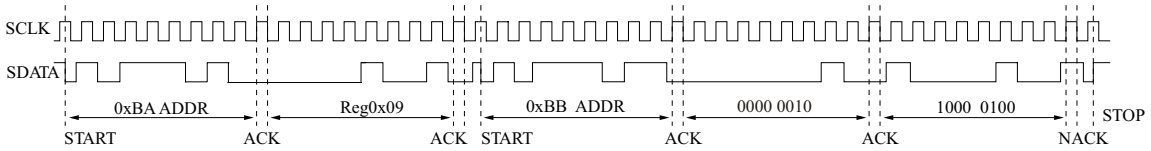


Figure 7.3: *Timing diagram for a typical read cycle [Micron, 2005, p. 34].*

The communication between the ARM and the image sensor is primarily written in assembly code and is divided into the following six functions.

1. start_write: Sends a start bit and sets the image sensor in write mode and reads the acknowledge bit.
2. start_read: Sends a start bit and sets the image sensor in read mode and reads the acknowledge bit.
3. load_register: Writes 8 bits of register address and reads the acknowledge bit.
4. load_setting: Writes 16 bits of data and reads the acknowledge bits.
5. read_setting: Reads 16 bits of data and sends the acknowledge bits.
6. stop_bit: Sends a stop bit.

The code is divided into these functions, because this makes the code reusable due to the similarities in the read and write sequences. The separation of the serial communication is possible, because there is no requirements concerning the slowest data transfer rate.

7.3 SRAM Software

The SRAM software is executed on the ARM during the Read-out sequence. It performs these of the actions hold by the ARM in Section 5.4.6, page 66. To understand why these actions are to be performed, refer to this section. The actions are:

1. ARM asserts PIO1 to switch the FPGA into Read-out mode.
2. ARM pulses PIO3 HIGH to reset the value of the counter in the FPGA.
3. ARM pulses PIO4 HIGH to trigger the image sensor.
4. ARM starts monitoring PIO2 and waits for it to be set HIGH.
5. When the ARM has detected that PIO2 has been set LOW, it sets PIO1 LOW to switch the FPGA back to Pass-through mode.

During the Read-out mode the FPGA handles DMA requiring the ARM to tri-state its data bus to avoid bus conflict. Tri-stating the data bus can be obtained by resetting the ARM, but since it must provide a 10 MHz clock signal to the image sensor during the Read-out sequence, this is not an option. Besides, the ARM must be able to detect when read-out of image data has ended. Therefore, a software image in internal SRAM is executed during the Read-out sequence.

Bus conflicts will then not be an issue since: "during any internal access to registers or internal ram, the external bus will stay in its inactive state, there won't be any toggling." [Filippi, 2006].

The interrupt handler and scheduler of eCos are disabled before executing the SRAM software. This is done to ensure that no communication is made between the ARM and the external memory, at which the data bus is tri-stated. The hibernation status of eCos means that communication with CDH is not possible.

Executing the SRAM code is obtained by moving the program counter to the location of the SRAM code while saving the program counter in a register. To exit the SRAM code the program counter is moved to the location one instruction after the jump address. Besides, it is ensured that the pipeline at all time contains valid instructions and that the interrupt handler and scheduler of eCos are enabled once again.

7.4 Suspending the Image Sensor

After the Read-out sequence is complete the image sensor is suspended to save power. This is done by suspend_image_sensor().

1. **ARM negates Chip Enable in the register of the image sensor.**
   Chip Enable is negated by clearing the setting in register 0x07 bit 1 of the image sensor, which stops the readout of the image sensor and powers down the analog circuitry of the image sensor.

2. **ARM deactivates the image sensor by setting PIO0 HIGH.**
   PIO0 of the ARM asserts the standby pin of image sensor and makes the image sensor power down.

3. **ARM disables the IS-CLK clock output.**
   The clock output on T0TIOA0 is no longer needed and is therefore disabled.

7.5 Color Interpolation

The chosen image sensor, Micron MT9T001, possesses a color filter array (CFA) called a Bayer filter [Micron, 2005, p. 1]. To get a fullcolor image from the raw image an interpolation has to be made. The description of this takes place in this section.

An image sensor uses CFAs to capture color images, because the photodiodes in the image sensor are only capable of measuring light intensity, not colors. To capture a color image, CFAs are used to capture different colors at different locations on the image sensor. The color filter makes this happen by only letting a single color through, i.e. letting a certain waveband through.

The chosen image sensor uses a Bayer filter, which is an array of primary colors: Red, green and blue (R, G and B in short). The color filter is arranged as shown in Figure 7.4. The Bayer filter has twice as many filters letting green light through than filters letting red and blue light through, because the human eye has a greater resolving power with green light than with red and blue light [Wikipedia, 2006b]. In short this means that the human eye is better to distinct between details of the shade green than any other color.

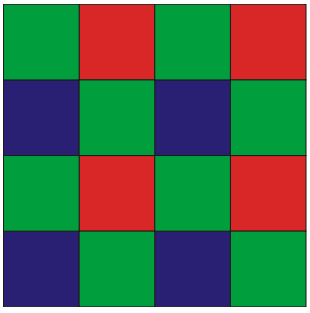


Figure 7.4: *A Bayer filter array of color filters.*

The chosen image sensor, Micron MT9T001, has quantum efficiency as shown in Figure 7.5, which implies that an IR-filter is necessary to prevent infrared light to be picked up by the image sensor. A full color image is wanted and thereby as so the full color image has to be interpolated

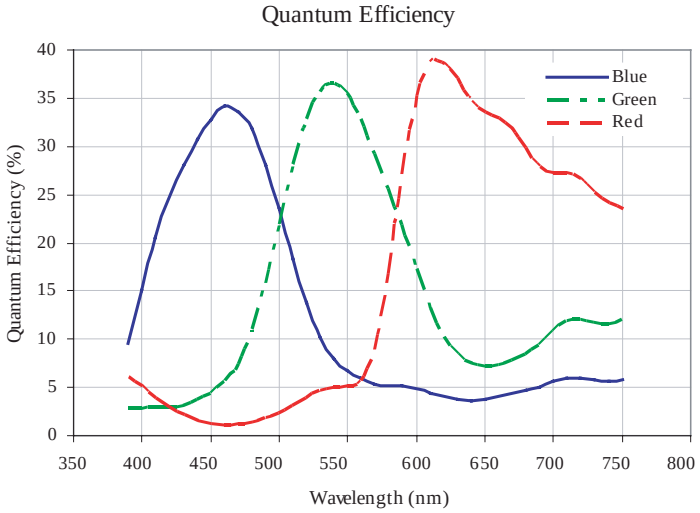


Figure 7.5: *Shows the quantum efficiency for the image sensor. The diagram is not in scale [Micron, 2005, p. 38].*

from the image of 8 bit raw Bayer pixels. This process is called demosaicing and will be discussed in the next section.

7.5.1 Demosaicing

The demosaicing process can be accomplished by using different algorithms of varying complexity depending on the desired image quality and number of iterations. There are a couple of problem areas in demosaicing a Bayer image.

One of the biggest challenges is to estimate the edges of the image properly. Here, the interpolation of the pixels is generally worse than in the rest of the image, because less pixels are available to interpolate from. Another problem is rapid color changes, which can make the estimate of a pixel very bad and may result in a blurred image.

To be able to choose which type of interpolation should be used in this project, the simplest demosaicing algorithms will be discussed in this section. The simplest way of demosaicing a raw image is by using bilinear interpolation, and therefore this will be discussed in the next section [Henrique S. Malvar and Cutler, 2004].

Bilinear Interpolation

In bilinear interpolation the average value of the pixels surrounding the pixel being interpolated are calculated and becomes a token of the interpolated pixel. A model illustrating how interpolation is accomplished in different pixels is described in the following text.

When the green color is interpolated on the place of a original blue or red pixel, the average of the four green pixels surrounding the pixel becomes the green color in the pixel being interpolated, which is shown in Equation (7.1). The Bayer filter in which the pixels are interpolated is shown in Figure 7.6.

$$G_{i,j} = \frac{G_{i+1,j} + G_{i-1,j} + G_{i,j+1} + G_{i,j-1}}{4} \quad [\cdot] \tag{7.1}$$

When a red or blue pixel is interpolated on the place of an original green pixel, the color of the pixel being interpolated becomes the average of the two red or blue pixels next to the green pixel.

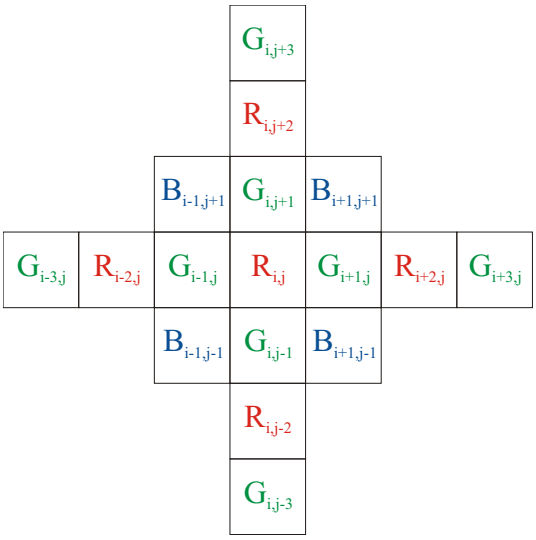


Figure 7.6: *A section of pixels in a Bayer image.*

An example of this is shown in Equation (7.2).

$$R_{i+1,j} = \frac{R_{i,j} + R_{i+2,j}}{2} \qquad [\cdot] \tag{7.2}$$

When a red pixel is interpolated on the place of an original blue pixel or a blue pixel is interpolated on the place of a original red pixel, the average value of the color of the four pixels in the corners of the pixel being interpolated become the color of the pixel being interpolated. An example of this is shown in Equation (7.3).

$$B_{i,j} = \frac{B_{i-1,j+1} + B_{i-1,j-1} + B_{i+1,j+1} + B_{i+1,j-1}}{4} \qquad [\cdot] \tag{7.3}$$

Bilinear interpolation uses two calculations on every pixel estimating the color information from the two missing colors. To improve the quality of the bilinear interpolation a median filter can be included in the algorithm, or an improved interpolation type can be used. An interpolation method that uses but improves bilinear interpolation is the constant hue-based interpolation. This method is described in the next section.

Constant Hue-Based Interpolation

Constant hue-based interpolation was one of the first demosaicing methods used in commercial camera systems [Ramanath et al., 2002, p. 2]. Constant hue-based interpolation first interpolates the green color in every pixel using bilinear interpolation or an other interpolation method. Afterward, the interpolation of the blue and red colors rely on the assumption that the hue, i.e. color ratio, changes smoothly, and therefore is equal for pixels in a small area. This means that the color ratio of two neighbor pixels is the same for all colors. This implies that for a image with constant hue $R_{i,j}/R_{i+1,j} = G_{i,j}/G_{i+1,j}$ if $B_{i,j}/B_{i+1,j} = G_{i,j}/G_{i+1,j}$. This makes it possible to calculate $R_{i+1,j}$ as shown in Equation (7.4).

$$R_{i+1,j} = \frac{R_{i,j}}{G_{i,j}} \cdot G_{i+1,j} \qquad [\cdot] \tag{7.4}$$

The output of image sensors is often logarithmic. This means that the logarithm has to be taken of the ratio in Equation (7.4). For a logarithmic image sensor, $R_{k,l}$ can be calculated as shown in Equation (7.5).

$$R_{i+1,j} = 10^{\log(R_{i,j}) - \log\left(\frac{G_{i,j}}{G_{i+1,j}}\right)} \qquad [\cdot] \tag{7.5}$$

If the interpolation should be further improved an interpolation method using edge detection can be used. One of those is gradient based interpolation, which is explained in the next section.

Gradient Based Interpolation

Gradient based interpolation uses edge detection to interpolate raw images. The principle of the interpolation is to detect an edge on behalf of the known pixels in the area, and when an edge is detected the pixel being interpolated is determined on behalf of either the known pixels in the vertical or horizontal direction. This is done to avoid blurring the edges. An example of this is how to determine the green color in pixel (i, j), i.e. $G_{i,j}$. To know which pixels to make the interpolation on behalf of, the horizontal gradient, α, is calculated from the red pixels as shown in Equation (7.6) [Popescu and Farid, 2005, p. 4].

$$\alpha_{i,j} = \left| \frac{R_{i-2,j} + R_{i+2,j}}{2} - R_{i,j} \right| \quad [\cdot] \tag{7.6}$$

Then the vertical gradient, β, is calculated as shown in Equation (7.7).

$$\beta_{i,j} = \left| \frac{R_{i,j-2} + R_{i,j+2}}{2} - R_{i,j} \right| \quad [\cdot] \tag{7.7}$$

The next step is to choose the green pixels to interpolate the pixel on behalf of. There are three different situations and therefore three different equations to interpolate the green pixel (i, j) from. The background for the choice of gradient is that over an edge changes the color a lot, this means that not only the red color changes. When $\alpha < \beta$ the color is interpolated as shown in Equation (7.8).

$$G_{i,j} = \frac{G_{i-1,j} + G_{i+1,j}}{2} \quad [\cdot] \tag{7.8}$$

When $\alpha > \beta$ the color is interpolated as shown in Equation (7.9).

$$G_{i,j} = \frac{G_{i,j-1} + G_{i,j+1}}{2} \quad [\cdot] \tag{7.9}$$

When $\alpha = \beta$ the color is interpolated as shown in Equation (7.10).

$$G_{i,j} = \frac{G_{i-1,j} + G_{i+1,j} + G_{i,j-1} + G_{i,j+1}}{4} \quad [\cdot] \tag{7.10}$$

This interpolation is done to find all the green pixels. Then the red and blue pixels are interpolated from the difference between the red or blue color and the green color. The color of the pixel is calculated differently depending on its position. When for example the blue color is calculated on the place of a original red pixel, (i, j), Equation (7.11) is used.

$$\begin{aligned} B_{i,j} = &\frac{(B_{i-1,j-1} - G_{i-1,j-1}) + (B_{i-1,j+1} - G_{i-1,j+1})}{4} \\ &+ \frac{(B_{i+1,j-1} - G_{i+1,j-1}) + (B_{i+1,j+1} - G_{i+1,j+1})}{4} + G_{i,j} \quad [\cdot] \end{aligned} \tag{7.11}$$

To determine red or blue colors on the place of a original green pixel one of the following two equations are used depending on the blue or red pixels surrounding the green pixel horizontally or vertically.

$$R_{i+1,j} = \frac{(R_{i,j} - G_{i,j}) + (R_{i+2,j} - G_{i+2,j})}{2} + G_{i+1,j} \quad [\cdot] \tag{7.12}$$

$$R_{i,j+1} = \frac{(R_{i,j} - G_{i,j}) + (R_{i,j+2} - G_{i,j+2})}{2} + G_{i,j+1} \quad [\cdot] \tag{7.13}$$

This interpolation uses the green color to interpolate all pixels. This has the advantage of maintaining color information in all pixels. When implementing this interpolation method a threshold is of course necessary on the comparison of the gradients to avoid detecting an edge everywhere in the image.

7.5.2 Choice of Interpolation Method

Bilinear interpolation is a simple and fast interpolation method and is therefore chosen. At the start of the project it was considered to use an algorithm from Devitech ApS, but this algorithm was never handed over to the project group due to lack of time from both Devitech ApS and the project group.

7.6 Resizing Image

When creating a thumbnail no interpolation is performed, as for the full image. This result from the fact that there is no need to predict missing pixels as the image has to be smaller than the original image. Instead the image is resized by resize_image().

If the image has to be reduced by a factor of two in height and width, it corresponds to taking each two by two pixel block and reducing it to one pixel. This reduction can be performed by simply using the red and blue values as is and taking the mean of the two green pixels in the block. If the image has to be reduced even further a technique called binning is used. This is done by taking the mean of all the blue pixels in a block and using that value as a single blue pixel, and likewise for the red and green pixels. If for instance the image has to be resized by a factor four in height and width, there are 4 red and 4 blue pixels in the block and 8 green pixels that has to form the value for a single pixel.

After the actual resize is done, the image goes through the same compression process as a full image. It can be chosen to do the compression with a little more loss to reduce the data size even further, as the image quality of the thumbnail does not have to be very good.

7.7 Image Compression

When images have to be sent from a satellite to Earth there are some important aspects that have to be taken into consideration. The most important aspect is the size of the image data. The available bandwidth of the spacelink is limited and as a consequence there is no point in having images with a very large data size, because it will take a considerable amount of time to download the image, if it will ever succeed. In addition, the transmission will also require a lot of energy. An 1.5 Mpixel image with 8 bit per pixel has a size of 1.5 MB before interpolation, and according to **§FUNC5** the maximum size of the image must be 256 kB. If it is chosen later to use the full capacity of 3 Mpixel of the image sensor, the amount of image data will be even bigger. If the image is interpolated in the satellite before sending the image to Earth, the size is even larger; an interpolated image with 3 channels with each 8 bit color data per pixel will have a size of 4.5 MB. This indicates the necessity of compressing the image before sending it to Earth.

When compressing an image there is a huge number of different techniques that can be used to minimize the data size of the image. At an overall basis the compression methods can be separated into two different categories, namely lossless and lossy methods. When using the lossless compression methods the original image can be restored from the compressed image, exactly as it was. An image compressed with a lossy method cannot be restored exactly to the original. In generally, the more an image has been compressed, the further the restored image will be from the original. Higher compression ratio results in an image that differs more from the original. When compressing an image it is important how big the entropy in the image is, because smaller entropy means easier to compress and smaller size of output.

The known image formats like JPEG, TIFF, PNG and GIF all use different compression methods; normally more than one technique is used in an image compression process. This is done because the compression factor is often larger when more techniques are combined than when just a single technique is used. The output from a compression, that is the compressed image, is often referred to as a bit stream.

In the following sections a number of basic compression techniques will be described, with reference to the choice of an image compression type for use in the camera system.

7.7.1 Color Channels of an Image

Under normal circumstances an image is described by the three primary colors, namely red, green and blue. The color channels, when combined, corresponds to a full color image as the human eye catches it. Normally, this separation into primary color channels is desired, but when an image has to be compressed it is not. The reason for this is most clearly visible by looking at an example. The picture in Figure 7.7 is shown in the three separate color channels in Figure 7.8, Figure 7.9 and

Figure 7.10. As can be seen on the images there are a lot of details in all of the channels, which is a huge disadvantage because more details occupies more space when the image is compressed.

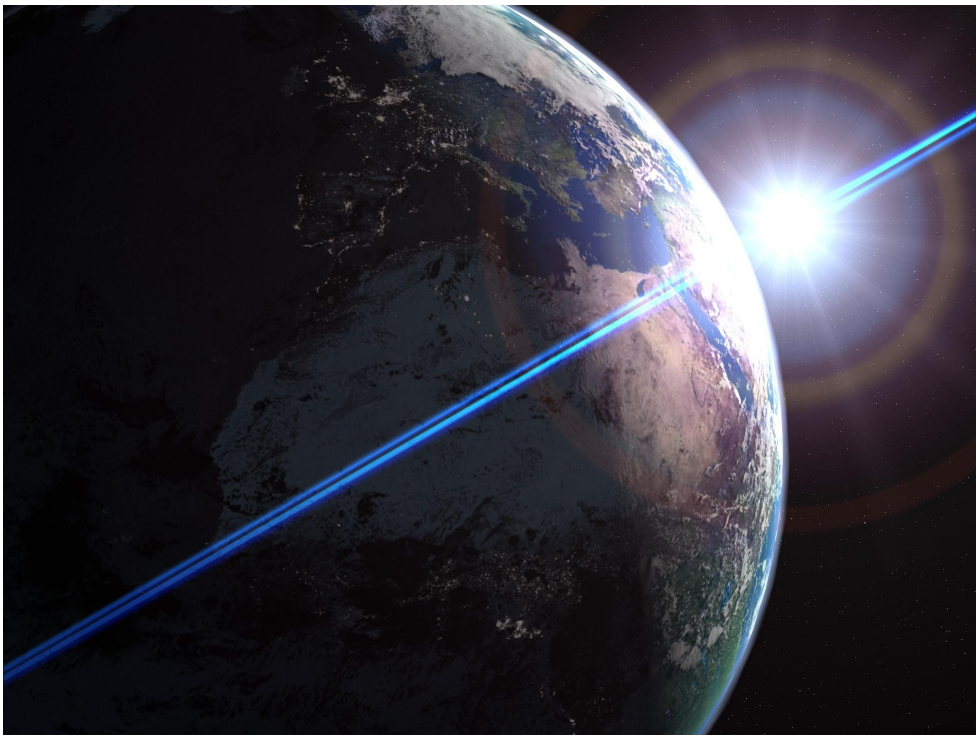


Figure 7.7: *The test image used in this section to show some image principals.*

To make an image easier to compress a lot of compression techniques converts the image into a different color space. A color space that is often used is YCbCr, where Y is the luma component (brightness) and Cb and Cr are the chroma components (color). The chroma components defines the difference between the colors blue (Cb) and red (Cr) and a reference color of the same luminous intensity, here it is Y. This color space has the advantage that the Y channel contains a lot of details, while the other two channels Cb and Cr in most cases does not have very much information in them, because the deviation is very small. In Figure 7.11, Figure 7.12 and Figure 7.13 the test image is converted into the YCbCr color space. It is easy to see that the Cb and Cr channels are very uniform.

Even though, the YCbCr channels seem to hold less information than the RGB channels, there is a very easy and almost lossless conversion both ways between the two color spaces. The reason for this to be only almost lossless is due to rounding errors, but it can only cause an error of maximum one shade of color pr. channel in a pixel. The conversion from RGB to YCbCr can be done according to the following equations [Wikipedia, 2006h].

$$\mathrm{Y_{m,n}} = \mathrm{Kr} \cdot \mathrm{R_{m,n}} + (1 - \mathrm{Kr} - \mathrm{Kb})\mathrm{G_{m,n}} + \mathrm{Kb} \cdot \mathrm{B_{m,n}} \quad [\cdot] \tag{7.14}$$

$$\mathrm{Cb_{m,n}} = \frac{0.5}{1 - \mathrm{Kb}}(\mathrm{B_{m,n}} - \mathrm{Y_{m,n}}) \quad [\cdot] \tag{7.15}$$

$$\mathrm{Cr_{m,n}} = \frac{0.5}{1 - \mathrm{Kr}}(\mathrm{R_{m,n}} - \mathrm{Y_{m,n}}) \quad [\cdot] \tag{7.16}$$

where:

m and n form the coordinates in the image $[\cdot]$.

Y, Cb and Cr are the color channels in the YCbCr color space $[\cdot]$.

R, G and B are the color channels in the RGB color space $[\cdot]$.

Kb is a constant set according to the specific YCbCr standard, mostly set to 0.114 $[\cdot]$.

Kr is a constant set according to the specific YCbCr standard, mostly set to 0.299 $[\cdot]$.

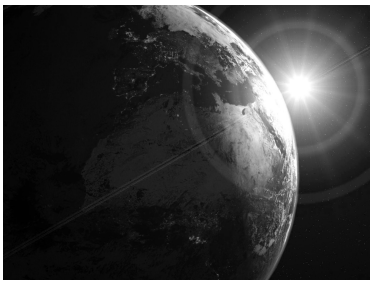


Figure 7.8: *The red channel of the color image on* Figure 7.7.

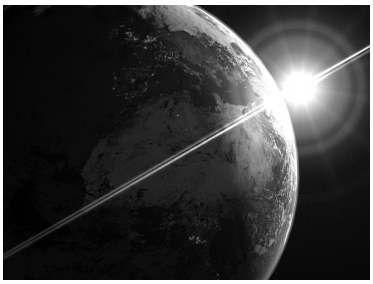


Figure 7.9: *The green channel of the color image on* Figure 7.7.

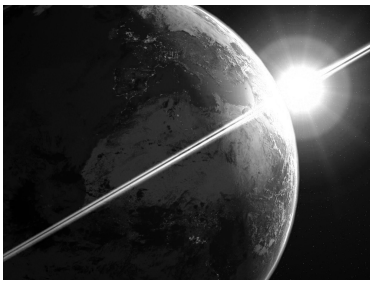


Figure 7.10: *The blue channel of the color image on* Figure 7.7.

Figure 7.11: *The Y channel of* Figure 7.7 *computed from the RGB channels.*

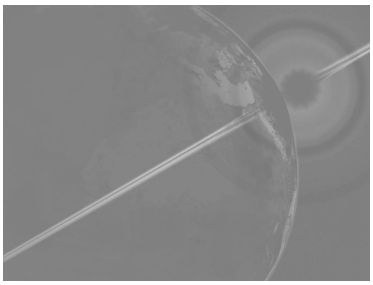


Figure 7.12: *The Cb channel of* Figure 7.7 *computed from the RGB channels.*

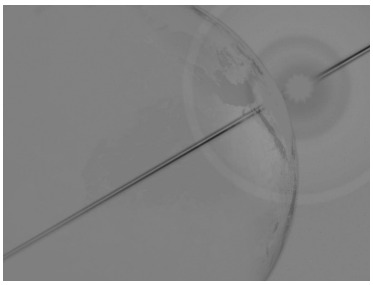


Figure 7.13: *The Cr channel of* Figure 7.7 *computed from the RGB channels.*

From theses equations it is also easy to derive the equations for the conversion from YCbCr to RGB. The red channel can be calculated from the Y channel and the Cr channel. It is easiest done by isolating R in Equation (7.16).

$$\mathrm{R_{m,n}} = \mathrm{Y_{m,n}} + 2(1-\mathrm{Kr})\mathrm{Cr_{m,n}} \quad [\,\cdot\,] \tag{7.17}$$

The same thing can be done for the blue channel by isolating B in Equation (7.15).

$$\mathrm{B_{m,n}} = \mathrm{Y_{m,n}} + 2(1-\mathrm{Kb})\mathrm{Cb_{m,n}} \quad [\,\cdot\,] \tag{7.18}$$

The green channel is a bit more tricky. The first step is to isolate G in Equation (7.14). The second step is to insert the expressions for R and B from Equation (7.17) and Equation (7.18). The last step is sorting it and the result is shown in Equation (7.19).

$$\begin{aligned}
\mathrm{G_{m,n}} &= \frac{\mathrm{Y_{m,n}} - \mathrm{Kr}\cdot\mathrm{R_{m,n}} - \mathrm{Kb}\cdot\mathrm{B_{m,n}}}{1-\mathrm{Kr}-\mathrm{Kb}} \\
\mathrm{G_{m,n}} &= \frac{\mathrm{Y_{m,n}} - \mathrm{Kr}(\mathrm{Y_{m,n}} + 2(1-\mathrm{Kr})\mathrm{Cr_{m,n}}) - \mathrm{Kb}(\mathrm{Y_{m,n}} + 2(1-\mathrm{Kb})\mathrm{Cb_{m,n}})}{1-\mathrm{Kr}-\mathrm{Kb}} \\
\mathrm{G_{m,n}} &= \mathrm{Y_{m,n}} - \frac{2\mathrm{Kr}(1-\mathrm{Kr})}{1-\mathrm{Kr}-\mathrm{Kb}}\mathrm{Cr_{m,n}} - \frac{2\mathrm{Kb}(1-\mathrm{Kb})}{1-\mathrm{Kr}-\mathrm{Kb}}\mathrm{Cb_{m,n}} \quad [\,\cdot\,]
\end{aligned} \tag{7.19}$$

Often when compressing an image, only the values between 0 and 255 are wanted; in this conversion the Cb and Cr channels have values from -128 to +127, and because of that, an offset of 128 is often used.

Because the Cb and Cr channels hold so little information they are sometimes resized to e.g. half of the original pixel size before compression. This is an additional technique that is used in e.g. JPEG, as it makes the amount of data to compress even smaller.

7.7.2 Block Prediction

The image is in a lot of compression methods divided into 8×8 pixels blocks and then processed block by block. The block prediction method takes these blocks one at a time, from left to right and from top to bottom, and makes a prediction of the content with the previous blocks as basis. The prediction is done from the information in the 25 pixels marked with gray in Figure 7.14. The prediction algorithm finds the best prediction among a number of predefined prediction mode possibilities. The one chosen as the best prediction is the one closest to the original image. The predicted modes can be encoded into the output bit stream in only a few bits, and then the only additional information needed to recreate the original image is the difference between the predicted blocks and the original image. This step of the compression is shown in pseudo-code in Equation (7.20).

$$\begin{aligned} \text{compressed} &= \text{compress}([\text{original-predicted}] + \text{predicted}) \\ \text{original} &= \text{decompress}(\text{compressed}) \end{aligned} \tag{7.20}$$

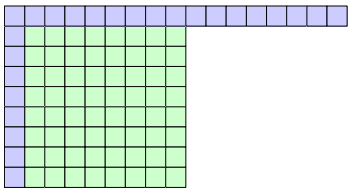


Figure 7.14: *An overview of the important pixels when making a block prediction. The gray pixels are the already known pixels and the green pixels are those about to be predicted.*

As an example the possible prediction modes in the H.264 standard are illustrated in Figure 7.15 to Figure 7.23 [van Bilsen, 2004b]. Even though these modes are the ones used in the H.264 standard there are a lot of other standards that use other prediction modes, JPEG uses e.g. 64 different modes.

The advantage of making predictions is that the prediction modes can be a part of the compressed file, and only one bit information is needed if the prediction mode is the same as the previous, and 4 bits if it is another mode. One of the 4 bit tells that it is not the same as the previous and 3 bits tells about the mode number. Notice that it is already known that the mode is another than the mode of the previous block, so only 8 modes remain, which corresponds to 3 bits.

When the predicted blocks are subtracted from the original image blocks the result is an image with a much smaller entropy than the original image. Because it has a small entropy all of the coefficient values are very similar, and small, and are easier to compress making the compressed image smaller.

7.7.3 Discrete Cosine Transform and Quantization

Discrete Cosine Transform (DCT) is a technique that is used to transform the values for the pixels in an 8×8 pixel block into coefficients to cosine functions with increasing frequency, also called a Fourier series. This operation can be performed at one row at a time and is then called 1-dimensional. It is also possible to perform the transformation as a 2-dimensional operation, taking the whole block into consideration and not only a single row.

When a DCT is performed it results in creation of a number of coefficients corresponding to the number of original pixels. Unlike the original intensity values the DCT coefficients can be both positive and negative numbers. The important thing is that in general all the last DCT coefficients are very small and thereby insignificant. The image can be restored from the first coefficients without noticeable loss. When the high frequency components are ignored it corresponds to a low pass filtration in the frequency domain. An example of a DCT transform of a given set of pixels is shown in Figure 7.24. The DCT coefficients shown in the figure will be used in Figure 7.25 to reconstruct the original color bar.

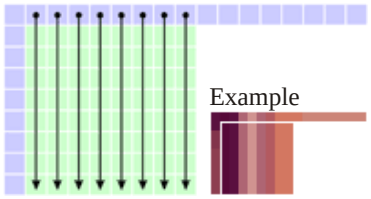


Figure 7.15: *0. Vertical - The upper pixels are extrapolated vertically.*

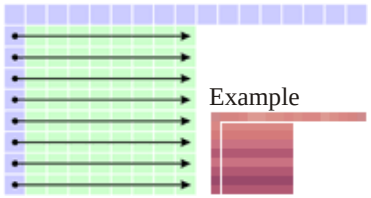


Figure 7.16: *1. Horizontal - The left pixels are extrapolated horizontally.*

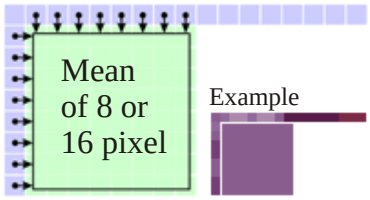


Figure 7.17: *2. DC - The mean of the available upper and left samples is used for the entire block. If none are available (first block in the image), the value 128 is used (50 % gray).*

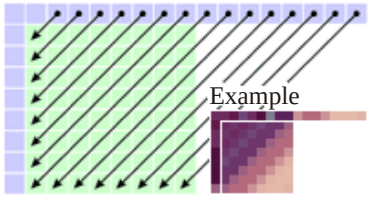


Figure 7.18: *3. Diagonal Down-Left - The pixels are interpolated at a 45° angle from the upper-right to the lower-left corner.*

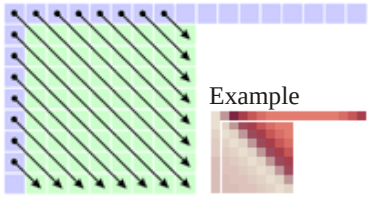


Figure 7.19: *4. Diagonal Down-Right - The pixels are interpolated at a 45° angle from the upper-left to the lower-right corner.*

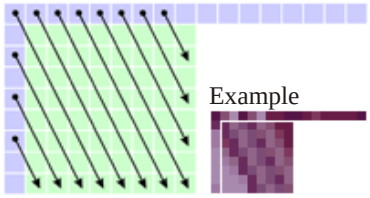


Figure 7.20: *5. Vertical-Right - The pixels are interpolated at a 26.6° angle from the upper-left corner to the lower edge at half the width.*

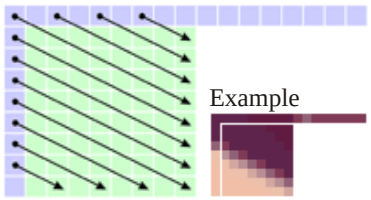


Figure 7.21: *6. Horizontal-Down - The pixels are interpolated at a 26.6° angle from the upper-left corner to the right edge at half the height.*

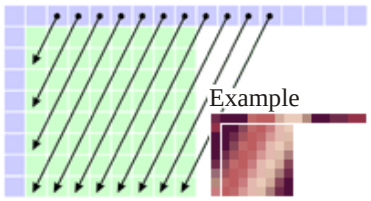


Figure 7.22: *7. Vertical-Left - The pixels are interpolated at a 26.6° angle from the upper-right corner to the lower edge at half the width.*

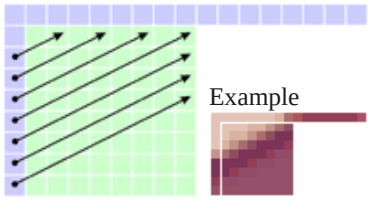


Figure 7.23: *8. Horizontal-Up - The pixels are interpolated at a 26.6° angle from the lower-left corner to the right edge at half the height. The bottom-right pixels are set to the bottom-most available pixel.*

Sample row	
Color value	121 58 61 113 171 200 226 246
DCT coefficients	61 -175 43 49 37 10 5 5

Figure 7.24: *An example of a DCT transform. The DCT coefficients do not have a direct connection to each of the individual original values, but there will always be the same number of DCT coefficients as there are original values.*

Each individual diagram in the reconstruction figure shows the reconstructed color bar with a different number of the DCT coefficients used. The red points are the values representing the colors in the original bar. The nearer the blue bar gets to the red points the more accurate the

reconstructed image is. It is clear to see the insignificance of the last coefficients by looking at the example of the reconstruction from the DCT coefficients. By throwing away the last coefficients the image can be, with only very limited loss of quality, compressed harder because a smaller number of coefficients have to be encoded.

It can be very useful to use quantization, which is a technique that divides the coefficients by a certain number and thereby making the numbers smaller. For small numbers not as many bits are needed to represent the number; again reducing the compressed image size. The cost of the compression is loss of image quality because the image coefficients cannot be restored accurate from the quantizised numbers. The bigger a number used for the quantization, the smaller the result will be, but at the same time the error will also be more comprehensive because of the rounding error. The quantization factor is often the one that is adjusted when choosing a compression level; this is the case in e.g. JPEG.

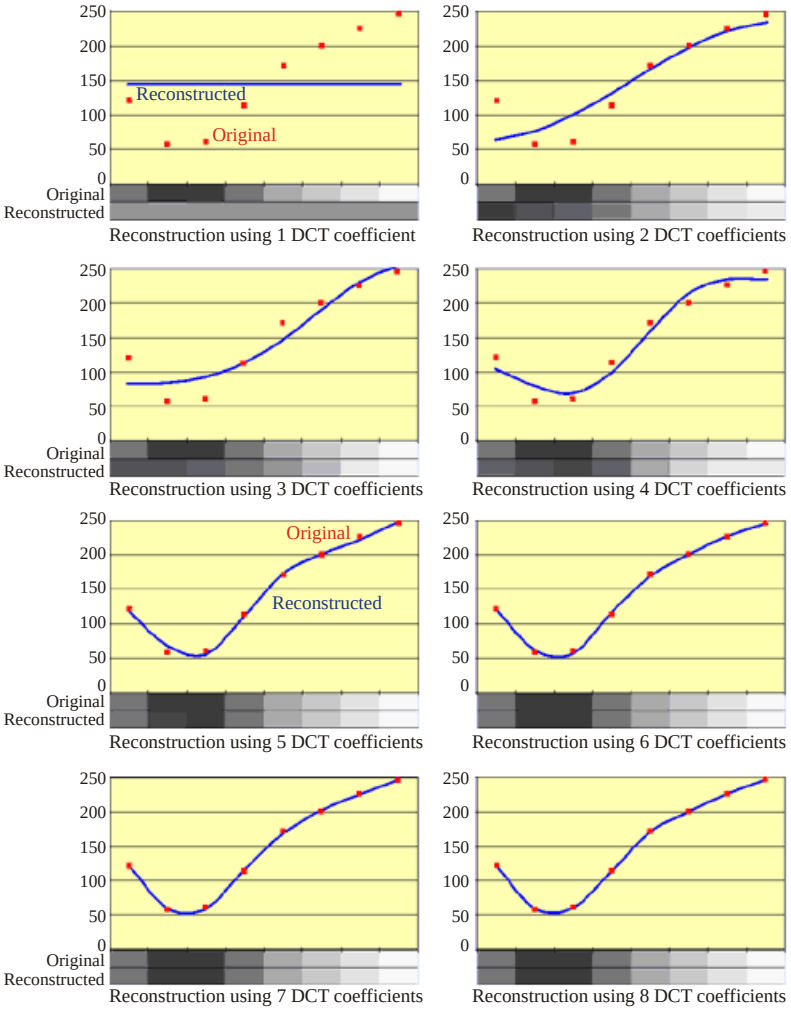


Figure 7.25: *An example of a reconstruction of an color bar from the DCT transform.*

7.7.4 Comparison and Choice of Image File Format

As can be seen in Table 7.1, the JPEG compression method has the advantage of making smallest files. The most obvious method to choose is therefore JPEG, but because it is a well defined standard the use of this method would require the download of a complete running code without the possibility of making changes. As a consequence, it has been chosen to take a starting point in another rather unknown format that allows a little more changes and gives a little challenge in implementing it. In this project a compression method starting from a method called AIC, Advanced Image Compression, is therefore chosen [van Bilsen, 2004a].

File format	Methods	Compression ratio	Notes
JPEG	Color space transformation Downsampling Discrete cosine transform Quantization Entropy coding	91.4 % - 99.3 %	(Lossy) Not good if the image is going to be edited and saved numerous times, because of loss in quality every time.
TIFF	(LZW compression)	0 % (Loseless) 71 % (Lossy/ compressed)	(Lossless/Lossy) Can hold several images or other data in one file by the use of "tags" in the file header.
PNG	DEFLATE Prediction	78.6 %	(Lossless)
GIF	LZW compression	85.9 % - 91 %	(Lossless if the image does not contain more than 256 colors.) Can only store 8 bit of information pr. pixel.

Table 7.1: *A comparison of some of the well known file formats. The size is found by saving the test image on* Figure 7.7 *in the given file format. The compression ratio is in proportion to an uncompressed raw image with a size of 9216 kB.*

AIC is a method that combines techniques from other compression formats, e.g. the block prediction technique used by JPEG. The method is very well described with algorithms for every step in the compression process. The original source code is written in Borland Delphi, and is thereby a good reference software, but since the code for this project is written in C, the original code can only be used as an inspiration on how to implement the algorithms. As it can be seen on Figure 7.26, the AIC compression is even very close to match the JPEG-2000 codec, which is an improvement of the JPEG codec, in quality vs. compression. At some bit rates AIC even outperforms the JPEG-2000 codec. The comparison is done with the well known test image Lena, which can be seen on Figure 7.27. Also on some other test images, especially photographic images, the AIC compression is close to match or even outperform the JPEG-2000 [van Bilsen, 2004a]. A consequence of choosing a non standard codec is that very few people already are capable of decompression the image. This is however not a problem since the images can be converted after they have been transferred to Earth; still it requires that a decompression algorithm is written.

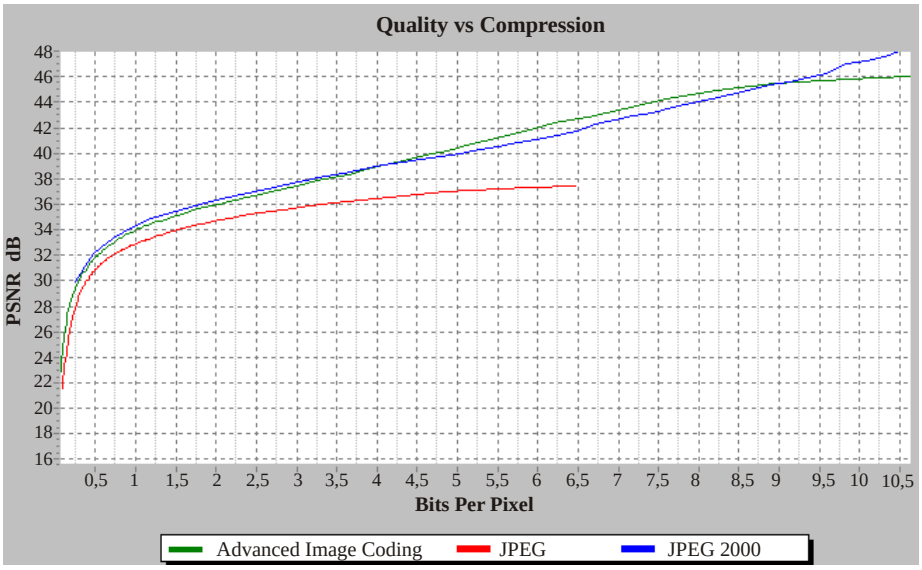


Figure 7.26: *The quality vs. compression for AIC, JPEG and JPEG-2000 [van Bilsen, 2004d].*

Figure 7.27: *The well known test image Lena [Rosenberg, 2001].*

The compression used in this project is done by performing the following steps:

1. The image is converted into the YCbCr color space as described in Section 7.7.1.
2. The block prediction method is used on the Y channel, which is the color channel holding most information.
3. The result of the block predictions are encoded into the output bit stream.
4. A discrete cosine transformation is performed on the pixel values as described in Section 7.7.3. The resulting coefficients of the DCT is also encoded into the output bit stream.
5. The differences between the predicted blocks and the original image forms a number of pixel values that are quantizised to make the values smaller. The quantization can be adjusted by setting the scale factor; a larger number results in smaller files but less accurate information.
6. An inverse discrete cosine transformation is made to produce the same pixel values as the decompression will have for the block predictions.
7. The steps 3 through 5 are followed for the Cb and Cr channels by using the same block prediction modes as for the Y channel, because the pattern in those channels usually look like the Y channel [van Bilsen, 2004c].

The original algorithms for the DCT used in AIC turned out to have a very low performance on the chosen ARM processor, since they use floating point calculations. This is not optimal, as it turned out that the ARM does not preform very well in floating point calculations. The DCT algorithm used by AIC come of the JPEG codec and by examining the JPEG reference software it was discovered that the JPEG codec has other possible DCT algorithms; the reference software can be downloaded from [Group, 1998]. It was chosen to implement one of the other algorithms for DCT from the JPEG codec that does not use floats.

During the compression, a 64 kB block of memory is used as temporary storage for the output bit stream since this corresponds to a FLASH block size where the image has to be saved for long term storage. When the block is full the content is written to FLASH and the memory reused for the rest of the bit stream. As there is a lot of heavy calculations in the compression process it is the most demanding job the ARM has to carry out. The complexity of the calculations also results in a time usage of several minutes to compress an image. The time usage is not optimal as more time results in a larger power consumption as the camera system has to be turned on and processing for a longer period.

Because of the time usage, it was decided to optimize the compression algorithms by using some of the more complex instructions provided by the ARM-instruction set. This meant that parts of the code were rewritten in assembly language for optimal exploitation of the ARM instruction set. The rewritten algorithms are the forward and inverse DCT, because they were very slow and and because both of them are run one time for each prediction block in every channel, meaning that they are run several thousand times during an image compression. This optimization resulted in a less time consuming compression algorithm, but it still takes several minutes to compress an image. The compression could be made even faster by rewriting more parts of the compression procedure in assembly, but due to lack of time this is not done within the time boundaries of this project.

8 Testing

This chapter contains tests performed on the prototype of the camera system. Block tests of hardware and module tests of software are carried out before a system integration test can be performed. This is done to make sure no errors occur in the blocks or the modules before they are implemented into processes. Through the test of the modules some of the requirements from the Requirement Specification are tested.

8.1 Block Tests of the FPGA

To ensure the FPGA works as desired it is necessary to perform simulations and circuit tests of the FPGA. It is necessary to define test specifications that will test all the functionality of the FPGA.

Several tests are defined and divided into different groups of functionality. The procedure for each test are as a explained in list of steps which must be carried out. Results are provided for each individual test in a test chart, while the discussion of the obtained results are gathered for an entire group of functionality.

8.1.1 Setup Used for Circuit Test/Verification

During tests, the FPGA will be isolated from the remaining camera system by removing the jumpers.

Instead of using the image sensor to deliver a clock signal, a functiongenerator will be used to generate a clock input on IS-PIXCLK. The functiongenerator is set to deliver a 10 MHz square wave, which should equal the clock input from the image sensor.

Tests measuring delay, from a specific input signal is applied, to a specific output is present, will make use of a debouncing circuitry for connecting the input. This avoid rebounding of the input signal. This description of the circuit can be found on the CD in the folder Debouncing Circuitry.

The equipment used for testing the FPGA is present in Table 8.1.

Equipment	Brand and model	AAU #	Comment
Oscilloscope	Agilent 54621D	33838	10 MΩ // 15 pF analog probes
Multimeter	Fluke 37	08180	
Power Supply	Hameg HM7042	33885	
Functiongenerator	Philips PM5138	08688	

Table 8.1: *Equipment used for the tests of the hardware and the FPGA.*

8.1.2 Mode Independent Pass

The FPGA must be able to pass the PIO0 input signal to the IS-STB output, independently of mode and whether IS-PIXCLK is running or not.

Test procedure:

1. The IS-STB output is measured.
2. The initial state of PIO1 must be LOW and no clock input is applied to IS-PIXCLK. All other inputs are tired LOW.
3. The PIO0 input is switched between HIGH and LOW. It is verified the IS-STB output corresponds to the input.
4. The PIO1 input is set HIGH.
5. The PIO0 input is switched between HIGH and LOW. It is verified that the IS-STB output corresponds to the input.

6. The clock from the functiongenerator is applied to the IS-PIXCLK input.
7. The PIO0 input is switched between HIGH and LOW. It is verified that the IS-STB output corresponds to the input.
8. The PIO1 input is set LOW.
9. The PIO0 input is switched between HIGH and LOW. It is verified that the IS-STB output corresponds to the input.

A test chart are provided on Table 8.2.

State		**Verification**	
		IS-STB = PIO0?	
IS-PIXCLK running	PIO1	Simulation	Circuit
No	LOW	√	√
No	HIGH	√	√
Yes	HIGH	√	√
Yes	LOW	√	√
All other inputs = LOW, IS-PIXCLK = 10 MHz			

Table 8.2: *Test chart for testing:* ***The FPGA is able to pass the PIO0 input signal to the IS-STB output, independently of mode and whether IS-PIXCLK is running or not.***

Conclusion on the Test of the Mode Independent Pass

The FPGA was able to pass the PIO0 input to IS-STB in all tested situations and it should be possible for the ARM to control standby mode of the image sensor.

8.1.3 Pass-through Mode

During Pass-through mode tests, there will be no clock input on the IS-PIXCLK pin since it must be assured that this mode is functional without a clock.

Constant values for the Pass-through mode must correspond to the truth table in Table **??.**

Test procedure:

1. PIO1 is set LOW to set the FPGA in Pass-through mode.
2. All other inputs except for PIO4 are also set LOW. To ensure that IS-TRIG output is set due to the mode and not the PIO4 input, the PIO4 is set HIGH.
3. The state of the pins $\overline{\text{BYTE}}$, IS-$\overline{\text{OE}}$, IS-TRIG, Data 15, PIO2 are verified with a multimeter. To verify the high impedance state of Data 15 it was verified that the current through a 10 kΩ resistor was able to pull-up the output voltage to 3.3 V, when it was connected to V_{CC} and pull down the voltage to 0 V, when connected to GND.

A test chart are provided on Table 8.3.

Output connection	$\overline{\text{BYTE}}$	IS-$\overline{\text{OE}}$	IS-TRIG	Data 15	PIO2
Expected	HIGH	HIGH	LOW	Z	LOW
Simulation	√	√	√	√	√
Circuit	√	√	√	√	√
PIO1 = LOW, PIO4 = HIGH, IS-FRAME = HIGH, All other inputs = LOW					

Table 8.3: *Test chart for testing:* ***Constant values for the Pass-through mode are corresponding to the truth table in*** Table **??.**

Signals passed through the FPGA must be equal to the input signals and be routed between the correct pins.

Test procedure:

1. PIO1 is set LOW to set the FPGA in Pass-through mode.

2. The state of the pins $\overline{\text{CS}}$[0:3], Address[0:20], $\overline{\text{OE}}$, $\overline{\text{WE}}$, $\overline{\text{LB}}$, $\overline{\text{UB}}$ are verified with a set of test vectors as input. It will not be possible to test all possible combinations due to time restriction. The test vectors are shown in the test chart on Table 8.4.

Input	**Connection**						**Verification** Outputs = inputs?	
	Address[0:20]	$\overline{\text{CS}}$[0:3]	$\overline{\text{OE}}$	$\overline{\text{WE}}$	$\overline{\text{LB}}$	$\overline{\text{UB}}$	Sim.	Cir.
1	0000 0000 0000 0000 0000 0	0000	0	0	0	0	√	√
2	1111 1111 1111 1111 1111 1	1111	1	1	1	1	√	√
3	1010 1010 1010 1010 1010 1	1110	0	1	0	0	√	√
4	0101 0101 0101 0101 0101 0	1101	1	0	1	1	√	√
5	1111 0000 1111 0000 1111 0	1011	1	0	0	0	√	√
6	0000 1111 0000 1111 0000 1	0111	1	0	1	0	√	√
PIO1 = LOW, All other inputs = LOW								

Table 8.4: *Test chart for testing:* ***Signals passed through the FPGA are equal to the input signals and be routed between the correct pins****.*

Propagation delay of the signals passed through

This test only tests one connection at a time. However, due to limited testing time only a few connections will be tested. The measured propagation delay can be used to calculate wait states adjusted by the microcontroller.
Test procedure:

1. One of the output connections: $\overline{\text{CS}}$[0:3], Address[0:20], $\overline{\text{OE}}$, $\overline{\text{WE}}$, $\overline{\text{LB}}$, $\overline{\text{UB}}$ is chosen and connected to an oscilloscope.

2. The PIO1 must be LOW to set Pass-through mode.

3. The input signal which passes its value to the chosen output is connected to the debouncing circuit and the oscilloscope.

4. All other input signals are tied LOW.

5. The chosen input signal is switched HIGH.

6. The delay to the output switch to HIGH is measured.

7. The signal which must be tested for delay is set LOW.

8. The delay to the output switch to LOW is measured.

9. The test is repeated for other inputs and outputs.

A test chart are provided on Table 8.5.

Input and output pass connection	**Propagation delay**			
	Simulation		Circuit	
	t_{PLH}	t_{PHL}	t_{PLH}	t_{PHL}
Address 0	9.1 ns	9.1 ns	7.2 ns	9.3 ns
Address 8	8.4 ns	8.3 ns	6.6 ns	7.9 ns
Address 16	7.9 ns	7.9 ns	4.5 ns	6.5 ns
Address 20	5.0 ns	5.0 ns	8.6 ns	10.8 ns
Address[0:19]	9.1 ns	9.1 ns	N/A	N/A
$\overline{\text{CS}}0$	8.1 ns	8.1 ns	5.1 ns	6.2 ns
$\overline{\text{CS}}3$	9.7 ns	9.7 ns	5.2 ns	6.4 ns
$\overline{\text{WE}}$	9.1 ns	9.1 ns	3.5 ns	5.5 ns
$\overline{\text{OE}}$	8.9 ns	8.9 ns	6.5 ns	8.2 ns
$\overline{\text{UB}}$	8.8 ns	8.8 ns	5.1 ns	5.3 ns
$\overline{\text{LB}}$	9.2 ns	9.2 ns	5.3 ns	6.1 ns
t_{PLH} = LOW to HIGH, t_{PHL} = HIGH to LOW				
PIO1 = LOW, All other inputs = LOW				

Table 8.5: *Test chart for:* ***Propagation delay of the signals passed through.***

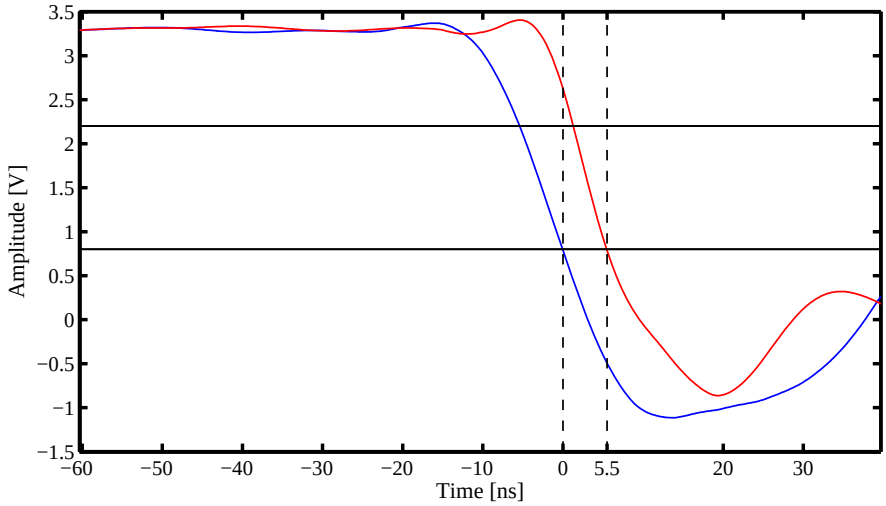


Figure 8.1: *The propagation delay is shown from the* $\overline{\text{WE}}$ *input is switch from HIGH to LOW till the* $\overline{\text{WE}}$ *output of the FPGA switches. The blue curve is the input voltage and the red is the output voltage.*

Conclusion of Pass-through Mode Tests

All simulations and tests of the circuit showed that the Pass-through mode worked properly and the measured values were as expected. The propagation delay were measured no higher than 10.8 ns. The only pins among the tested which were measured to have a higher propagation delay than in the simulation were the Address 0 and Address 20. It should be noted that a propagation delay is highly depended of the output load and the test setup only give a hint of the propagation delay when the memory is connected. The Address 20 output is configured to deliver a smaller output current since it is only loaded by a single input of the FLASH in contrast to other outputs which also are connected to RAM blocks. When simulations are performed they assume equal load capacitance on all outputs.

8.1.4 Read-out Mode

The first four tests of the Read-out mode are performed without a clock input on IS-PIXCLK, since they test functionality which must operate properly without a clock.

The Read-out mode must set constant output values corresponding to the truth table in Table **??.**

Test procedure:

1. PIO1 is set HIGH to set the FPGA in Read-out mode.
2. It is verified that the state of $\overline{\text{CS}}$[0:3], Address 20 (FLASH), $\overline{\text{BYTE}}$, $\overline{\text{UB}}$, $\overline{\text{LB}}$ and $\overline{\text{OE}}$ correspond to the truth table in Table **??**.

The test chart is shown on Table 8.6.

Output connection	$\overline{\text{CS}}$[0:2]	$\overline{\text{CS}}$3	A20 (FLASH)	$\overline{\text{BYTE}}$	$\overline{\text{UB}}$	$\overline{\text{LB}}$	$\overline{\text{OE}}$
Expected Value	HIGH	LOW	LOW	LOW	HIGH	LOW	HIGH
Simulation verification	√	√	√	√	√	√	√
Circuit verification	√	√	√	√	√	√	√
PIO0 = HIGH, IS-PIXCLK = LOW, All other inputs = LOW							

Table 8.6: *Test chart for testing:* ***The Read-out mode is setting constant output values corresponding to the truth table in*** Table **??**.

Read-out mode must pass image sensor signals correct.

Test procedure:

1. PIO1 is set HIGH to set the FPGA in Read-out mode.
2. It is verified that input of PIO4 is passed to IS-TRIG and the input of IS-FRAME is passed to PIO2. This is done by switching the inputs between HIGH and LOW and verifying that each output correspond to the input.

The test chart is shown on Table 8.7.

Pass		**Verification**	
Input	Output	Simulation	Circuit
IS-FRAME	PIO2	√	√
PIO4	IS-TRIG	√	√
PIO1 = HIGH, IS-PIXCLK = LOW, All other inputs = LOW			

Table 8.7: *Test chart for testing:* ***Read-out mode is passing image sensor signals correct.***

During Read-out mode the FPGA outputs must be immune to changes in memory control inputs from the ARM.

Test procedure:

1. The PIO1 must be set HIGH.
2. The outputs: Address[0:20], $\overline{\text{CS}}$[0:3], $\overline{\text{OE}}$, $\overline{\text{WE}}$, $\overline{\text{UB}}$ and $\overline{\text{LB}}$ are connected to an oscilloscope.
3. A set of test vectors are applied as signals on the inputs: Address[0:20] and $\overline{\text{CS}}$[0:3], $\overline{\text{OE}}$, $\overline{\text{WE}}$, $\overline{\text{UB}}$ and $\overline{\text{LB}}$.
4. It must be verified that the states of the outputs do not alter due to the different input vectors shown in Table 8.8.

	Connection						Verification	
Input	Address[0:20]	$\overline{CS}$[0:3]	$\overline{OE}$	$\overline{WE}$	$\overline{UB}$	$\overline{LB}$	Sim.	Cir.
Vector 1	0000 0000 0000 0000 0000 0	0000	0	0	0	0	√	√
Vector 2	1111 1111 1111 1111 1111 1	1111	1	1	1	1	√	√
Vector 3	1010 1010 1010 1010 1010 1	1110	0	1	0	0	√	√
Vector 4	0101 0101 0101 0101 0101 0	1101	1	0	1	1	√	√
Vector 5	1111 0000 1111 0000 1111 0	1011	1	0	0	0	√	√
Vector 6	0000 1111 0000 1111 0000 1	0111	1	0	1	0	√	√
PIO1 = HIGH, IS-PIXCLK = LOW, All other inputs = LOW								

Table 8.8: *Test chart for testing:* ***During Read-out mode the FPGA outputs are immune to changes in memory control inputs from the ARM.***

The following test utilizes a clock input, but to reduce the speed the clock input from the functiongenerator is not used. Instead, a clock is manually pulsed with a switch.

The counter must be able to count on the falling edge of IS-PIXCLK, and must only count when IS-LINE is asserted.

This test also verifies that the counter increments on the correct edge.
Test procedure:

1. The PIO1 must be set HIGH.
2. IS-PIXCLK is connected to a switch through the debouncing circuit.
3. The Address[0:20] outputs are connected to an oscilloscope.
4. Initial value of IS-LINE is HIGH and the initial value of PIXCLK is LOW.
5. The binary value on the Address bus is noted.
6. PIXCLK is switched HIGH.
7. It must be verified that the binary value on the Address bus has not incremented on the positive edge.
8. PIXCLK is switched LOW.
9. It must be verified that the binary value on the Address bus has incremented.
10. PIXCLK is switched LOW to HIGH and HIGH to LOW.
11. It must be verified that the binary value on the Address bus has incremented.
12. PIXCLK is switched LOW to HIGH and HIGH to LOW three times.
13. It must be verified that the binary value on the Address bus has incremented three values.
14. IS-LINE is switched LOW.
15. PIXCLK is switched LOW to HIGH and HIGH to LOW three times.
16. It must be verified that the binary value on the Address bus has not incremented.
17. IS-LINE is switched HIGH.
18. PIXCLK is switched LOW to HIGH and HIGH to LOW three times.
19. It must be verified that the binary value on the Address bus has incremented three values again.

A test chart are provided in Table 8.9.

Step	**Input connection**		**Binary value on Address bus**	**Verification**	
	IS-LINE	IS-PIXCLK			
-	HIGH	LOW	Initial value	Simulation	Circuit
1	HIGH	⌐	+ 0	√	√
2	HIGH	¬	+ 1	√	√
3	HIGH	1 × ⌐¬	+ 1	√	√
4	HIGH	3 × ⌐¬	+ 3	√	√
5	LOW	3 × ⌐¬	+ 0	√	√
6	HIGH	3 × ⌐¬	+ 3	√	√
Final value:			Initial value + 8	√	√
PIO1 = HIGH, All other inputs = LOW					

Table 8.9: *Test chart for testing:* ***The counter is able to count on the falling edge of IS-PIXCLK, and is only counting when IS-LINE is asserted.***

For remaining tests of the Read-out mode, a clock signal from a functiongenerator is connected to IS-PIXCLK. These tests rely on the counter being implemented as a loop counter.

The counter must be able to count properly to the limit of 2^{21}.

Test procedure:

1. The PIO1 and IS-LINE connections must be set HIGH.
2. The input clock on IS-PIXCLK must be running.
3. The Address[0:19] and A20/D15 connections are monitored.
4. It is verified that frequency of Address 0 (LSB) is 5 MHz and the duty cycle is 50%.
5. It is verified that frequency of next Address is half the frequency and the duty cycle is still 50%.
6. Step 5 is repeated up to A20/D15 which should output a frequency of 4.8 Hz and still have a duty cycle of 50%.

A test chart are provided in Table 8.10.

Output connection	**Expected frequency**	**Verification**			
		Duty cycle = 50% ± 1%			
		Frequency = expected frequency ± 1%			
		Simulation		Circuit	
		Frequency	Duty cycle	Frequency	Duty cycle
Address 0	5 MHz	√	√	√	√
Address 1	2.5 MHz	√	√	√	√
Address 2	1.25 MHz	√	√	√	√
Address 3	625 kHz	√	√	√	√
Address 4	312 kHz	√	√	√	√
Address 5	156 kHz	√	√	√	√
Address 6	78.1 kHz	√	√	√	√
Address 7	39.0 kHz	√	√	√	√
Address 8	19.5 kHz	√	√	√	√
Address 9	9.77 kHz	√	√	√	√
Address 10	4.88 kHz	N/A	N/A	√	√
Address 11	2.44 kHz	N/A	N/A	√	√
Address 12	1.22 kHz	N/A	N/A	√	√
Address 13	610 Hz	N/A	N/A	√	√
Address 14	305 Hz	N/A	N/A	√	√
Address 15	153 Hz	N/A	N/A	√	√
Address 16	76.4 Hz	N/A	N/A	√	√
Address 17	38.1 Hz	N/A	N/A	√	√
Address 18	19.1 Hz	N/A	N/A	√	√
Address 19	9.54 Hz	N/A	N/A	√	√
Address 20 / Data 15	4.77 Hz	N/A	N/A	√	√
PIO1 = HIGH, IS-LINE = HIGH, IS-PIXCLK = 10 MHz, All other inputs = LOW					

Table 8.10: *Test chart for testing:* ***The counter is able to count properly to the limit of*** 2^{21}.

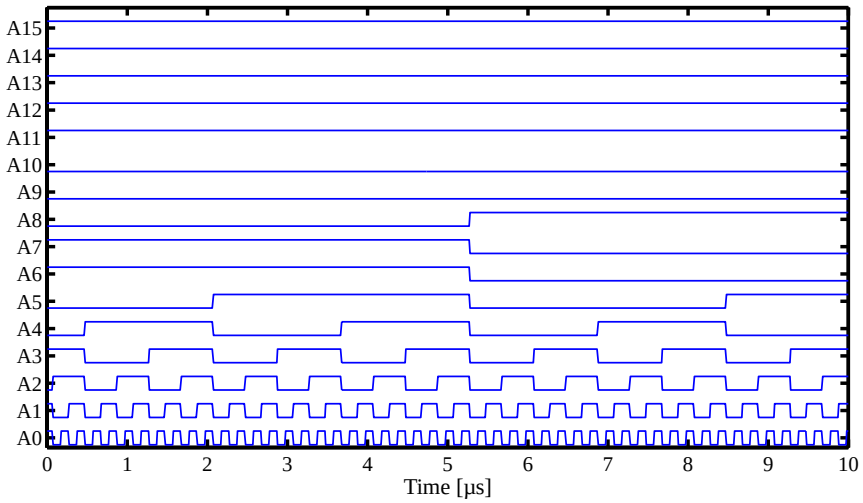


Figure 8.2: *The state of the Address [0:15] outputs while the counter for DMA is enabled. It can be seen how period time of the Address 1 output is doubled proportion the Address 0 output, the period time of the Address 2 output is likewise doubled in proportion to the Address 1 output, and so fort for the following Address outputs. The upper is Address outputs keep their state within the time measured.*

The counter outputs on the Address bus must reset to zero if PIO3 is pulsed HIGH.

Test procedure:

1. The PIO1 must be HIGH and the clock on IS-PIXCLK must be running.
2. The initial value of both IS-LINE and PIO3 must be LOW.
3. The Address[0:19] and A20/D15 outputs are monitored with an oscilloscope.
4. IS-LINE is pulsed HIGH for at least 100 ms.
5. It is verified that the binary address value on Address[0:19] and A20/D15 is not zero.
6. PIO3 is pulsed HIGH for at least 100 ns. This will ensure that at least one triggering edge from IS-PIXCLK occurs.
7. It is verified that the binary address value on Address[0:19] and A20/D15 has been reset to zero.

A test chart are provided in Table 8.11.

Address[0:19] and A20/D15 output	**Verification**	
	Simulation	Circuit
Binary value $\neq 0$	$\surd$	$\surd$
PIO3 = HIGH for min. 100 ns		
Binary value = 0	$\surd$	$\surd$
PIO1 = HIGH, IS-PIXCLK = 10 MHz, All other inputs = LOW		

Table 8.11: *Test chart for testing:* ***The counter outputs on the Address bus reset to zero if PIO3 is pulsed HIGH****.*

The counter output on the Address bus must not change before $\overline{\text{WE}}$ is negated.

The test will be performed on one connection of the Address bus at a time.
Test procedure:

1. The PIO1 and IS-LINE must be set HIGH.
2. The clock input on IS-PIXCLK must be running.
3. The $\overline{\text{WE}}$ output is connected to an analogues input on a oscilloscope.
4. One of the Address[0:19] and A20/D15 outputs are connected to the same oscilloscope.
5. It is verified that the voltage of $\overline{\text{WE}}$ has at least reached 2.2 V before the voltage on the Address output falls below 2.2 V from a stable value of 3.3 V or is increased above 0.8 V from a stable voltage of 0 V. The voltage range between 0.8 V and 2.2 V is not defined as a specific logic value and the write pulse must ended before the voltage of the Address output enters this range.

| **Output connection** | **Margin from $_\!\!_\!\!_\!\!

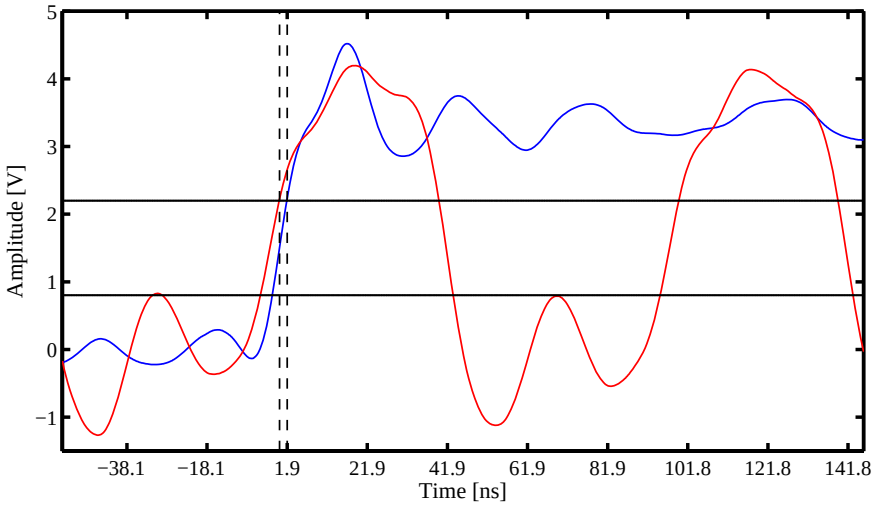


Figure 8.3: *Measurements verifying* $\overline{\text{WE}}$ *is negated before the Address bus become unstable. The red curve shows the voltage of the* $\overline{\text{WE}}$ *connection and the blue curve is the Address 16 output. The black horizontal lines are the logic threshold voltages of the RAM. As can be seen from the vertical cursor the* $\overline{\text{WE}}$ *output reach the 2.2 V HIGH threshold 1.9 ns before the Address 16 output. However, the requirement is not met since the Address 16 output voltage cross the lower threshold into the undefined range before* $\overline{\text{WE}}$ *reach 2.2 V.*

A write pulse must pulse $\overline{\text{WE}}$ LOW for at least 40 ns.

Test procedure:

1. The PIO1 and IS-LINE must be set HIGH.
2. The clock input on IS-PIXCLK must be running.
3. The $\overline{\text{WE}}$ connection is connected to an analogues input on a oscilloscope.
4. It is verified that each time $\overline{\text{WE}}$ is pulsed LOW, the voltage of $\overline{\text{WE}}$ is held below 0.8 V for at least 40 ns.

A test chart are provided in Table 8.13.

Output connection	**Width of pulse being LOW**	
	Simulation	Circuit
$\overline{\text{WE}}$	50 ns	55 ns
PIO1 = HIGH, IS-LINE = HIGH, IS-PIXCLK = 10 MHz, All other inputs = LOW		

Table 8.13: *Test chart for testing:* ***A write pulse does pulse*** $\overline{\text{WE}}$ ***LOW for at least 40 ns.***

Conclusion of Read-out Mode Tests

All simulations gave expected values and showed that timing constraints were kept. A few verifications of the higher Address outputs could not be performed in simulations since the counter variable started from zero and it would require the simulation to calculate a too large interval. In order to verify these Address outputs, the counter variable was changed to start at a higher value and it was verified the output changed at the correct value. The circuit tests showed that the correct outputs were set when the Read-out mode was activated and the FPGA was able to count properly; thereby the functionality of the Read-out mode were correct.

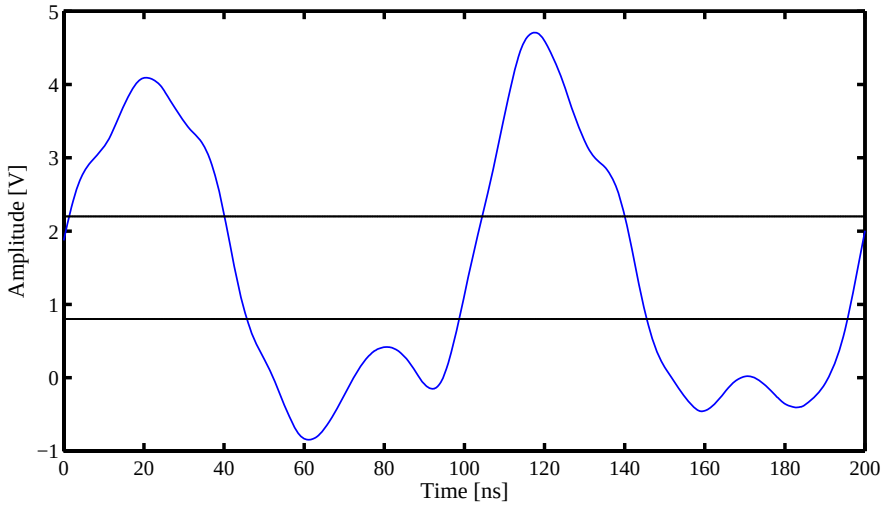


Figure 8.4: *The* $\overline{\text{WE}}$ *output LOW for 55 ns and thereby keeps the timing constraints which specifies that a write pulse must assert* $\overline{\text{WE}}$ *for at least 40 ns. The lower horizontal cursor are set to measure the lower threshold voltage of 0.8 V.*

The timing constraints from RAM of guarantying $\overline{\text{WE}}$ is negated before the value Address bus becomes unstable was however not kept, when FPGA was handling DMA in Read-out mode. Although the voltage measured on the Address outputs reached the undefined voltage range between 0.8 V and 2.2 V before the voltage of $\overline{\text{WE}}$ exceeded 2.2 V, the Address outputs never reached 2.2 V before $\overline{\text{WE}}$. Since the logic levels are expected to be equal for all connections on the same RAM, it is not expected to give any problems to use the FPGA for DMA. In the configuration of output pins in Section 5.4.9, page 74, the $\overline{\text{WE}}$ connection already have been selected to have a fast slew rate. However, it should still be possible to optimize the timing by further by adding an internal pull-up resistor or by increasing the output current. Alternatively, propagation delay of the counter variable can be extended with combinatorial logic.

8.2 Module Tests

The purpose of this section is to make white box tests of the six modules the camera system consists of. It is done to make sure that the modules perform as specified before they are united into processes.

8.2.1 Command Buffer

The command buffer should be tested to make sure it keeps up with **§FUNC10**. A separate test for this module is not performed, because the command buffer is dynamically allocated, which means that it can hold far more commands than required in **§FUNC10**. Furthermore, the command buffer is used throughout the test of the five other modules, meaning that if the other modules function the command buffer does too.

8.2.2 Setup Camera

The purpose of testing the module Setup Camera is to make sure that default settings can be fetched into current settings using cam_default() and that the validation of settings and registers using cam_setup(**data*, *length*) returns the expected values. The module is described in Section 6.5.2, page 87, and the flow is shown in Figure B.5 on page 181.

Test Description

The flow of testing Setup Camera has to be as shown below. The arguments to test register 0x02 and 0x23 is described separately, because register 0x02 is dependent of the setting written in register 0x23. Due to time constraints only a small selection of tests are performed as shown in Table 8.15.

1. Call the function cam_default().
2. Verify return value is DONE.
3. Call the function cam_setup(**data*, *length*) by the arguments shown in Table 8.14 and Table 8.15.
4. Verify return values.

Test Results

All four items in the test description were met and the return values are shown in Table 8.14 and Table 8.15.

Register 0x02	Register 0x23	Expected status	Verification
0x01	0x00	ERROR_INVALID_SETTINGS	√
0x02	0x00	DONE	√
0x04	0x30	DONE	√
0x1E	0x20	DONE	√

Table 8.14: *Shows the settings in resister 0x02 and 0x23 and the status messages.*

Register (Hex)	Setting (Binary)	Expected status	Verification
0x04	0000 0000 0000 0000	ERROR_INVALID_SETTINGS	√
0x04	0000 0000 0000 0001	DONE	√
0x04	0000 1000 0000 0000	ERROR_INVALID_SETTINGS	√
0x04	0000 0111 1111 1111	DONE	√
0x04	0000 0111 1111 1110	ERROR_INVALID_SETTINGS	√

Table 8.15: *Test of the test of validity of testing register and setting.*

Conclusion

The test of Setup Camera has run as expected and the function has returned the expected return values through the entire test. This means that the function performs as specified, which is to verify that only valid settings are transferred to the image sensor.

8.2.3 Capture Image

The purpose of testing Capture Image is to verify that the module captures an image in less than 1 sec. and that the interpolation, conversion to YCbCr, and compression runs as expected. Furthermore, the size of the thumbnail and compressed image have to be validated. Capture Image is explained in Section 6.5.2, page 88, and the flow of the function is shown in Figure B.7 on page 185.

Test Description

The module Capture Image should be tested by checking if the settings written to the image sensor are equal to the settings read from the registers of the image sensor. Furthermore, the image data should be tested after each step of the image manipulation to verify this process runs as expected. This implies that all error handling are not tested, but two requirements; **§FUNC5** and **§FUNC7** are tested. The requirements are written in Section 3.2, page 42. To test the two requirements the test specifications for these two requirements from Section 4.2, page 45, have been rewritten to one test, which is described in this section.

Before testing the module, FLAGS must be cleared to make sure that there are enough space in FLASH to save a raw, thumb, and compressed image, because FLAGS indicates if images are saved in FLASH as described in Section 6.4.1, page 80. The test flow must be as shown in Figure 8.5, where the square indicates a premise and a circle indicates a test. The premise and seven tests

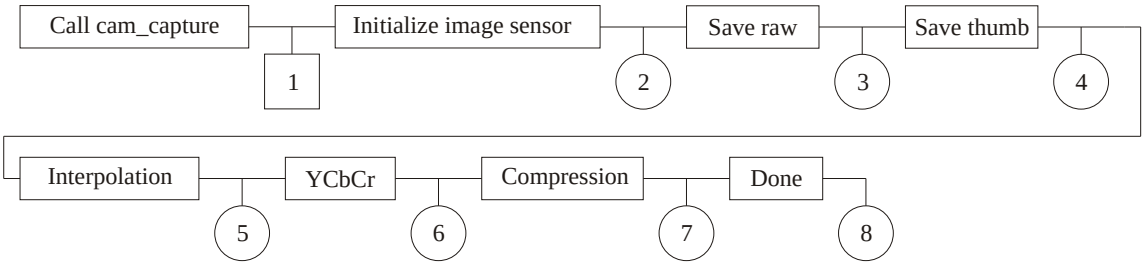


Figure 8.5: *Shows the flow of the test and the locations of the seven tests and one premise.*

must be as follows:

1. A raw image and a thumbnail will be created by calling the function by inserting True for both raw and thumb.
2. Read the settings from the image sensor and compare them with current settings in FLASH.
3. Raw image data is saved in FLASH and has to be downloaded to a PC with an USB to CAN converter at the end of the test.
4. Thumbnail is saved in FLASH and has to be downloaded to a PC with an USB to CAN converter at the end of the test to verify that the size of the thumbnail is no larger than 10 kB.
5. Interpolated image data has to be downloaded to a PC with an USB to CAN converter.
6. YCbCr image data has to be downloaded to a PC with an USB to CAN converter.
7. Compressed image is saved in FLASH and has to be downloaded to a PC with an USB to CAN converter at the end of the test to verify that the size of the compressed image is no larger than 256 kB.
8. Verify the return value equals DONE and that FLAGS have been set.

Conclusion

The test of Capture Image has not been accomplished, because the subfunctions included in Capture Image have not yet been integrated properly. All subfunctions except the image compression perform as expected and are integrated in Capture Image. The image compression does not work, because it is written to address the image in RAM blocks having subsequent address space. This is not the case on this microcontroller, because the RAM blocks used in this project have a size of 2 MB and the address page available on the ARM is 4 MB. The test of Capture Image is therefore not possible and will not be accomplished.

No images captured by the image sensor are shown in this test, because no images in focus were captured with the test lens. The downloaded data from the image sensor indicated that the image sensor is sensitive to light, but due to lack of time no further tests were performed. The module Capture Image is expected to be finished and tested before examination. The results of the tests are therefore expected to be published to the examination.

8.2.4 List Image

The module List Image should be tested to verify that the module can return an error message if no images exist and a pointer to a list if images exist. The flow of the module is shown in Figure B.8 on page 187.

Test Description

The module should be testet by testing the two different return values of the function. This is accomplished by following these seven items:

1. Clear all flags.
2. Call the function cam_list_images().
3. Verify that return value is NO_IMAGES_EXIST.
4. Set flags indicating three images are stored in FLASH.
5. Call the function cam_list_images().
6. Verify that return value is DONE.
7. Verify the list by checking *img_id*, *time*, *size*, and *type*.

Test Results

List Image can return both NO_IMAGES_EXIST, DONE, and make a list containing *img_id*, *time*, *size*, and *type*. This means that all verifications described in the test description were met.

Conclusion

The test of List Image turned out successfully, which means that the module performs as expected and can return a list if any images exist.

8.2.5 Delete Image

The purpose of testing List Image is to verify that the module Delete Image can delete an image if it exists, and that it can return an error message if no image exists for the *img_id* written as an argument to the function. The flow of the module is shown in Figure B.9 on page 189.

Test Description

The module should be testet by testing the two different return values of the function. This is accomplished by following these six items:

1. Set flags indicating a image with a corresponding thumb has been saved, then write 5 into *img_id*.
2. Call cam_delete_image(*img_id*), where *img_id* equals 5.
3. Verify return value is DONE.
4. Verify flags corresponding to *img_id* 5 has been deleted.
5. Call cam_delete_image(*img_id*), where *img_id* equals 5.
6. Verify return value is NO_SUCH_IMAGE.

Test Results

All verifications of the return values and flags of the test were met after following the six items above.

Conclusion

The test turned out successfully, meaning that the module Delete Image can delete an image and delete flags corresponding to the deleted image.

8.2.6 Send Image

The purpose of testing Send Image is to verify that the module can send the five different types of images in chunks of 10 kB each if such an image exists as described in **§FUNC6** else the module should return an error message. The flow of the module is shown in Figure B.10 on page 191.

Test Description

The module should be tested by following the five items bellow.

1. Capture an image and create thumbnail with *img_id* 3 and raw image.
2. Capture an image without a thumbnail with *img_id* 4.
3. Call cam_send_img with arguments specified in Table 8.16.
4. Verify return values are as shown in Table 8.17.
5. Verify image size and data, when return value is DONE and verify that the size of the received chunks do not exceed 10 kB.

img_id	***chunknumber***	***type***
2	X	0
3	X	1
4	X	1
4	X	0
3	300	3
3	10	3
3	X	2
3	13	5

Table 8.16: *Shows the arguments used when calling cam_send_img. X means don't care and is set to 0 during test.*

Test Results

The test of Send Image gave the return values shown in Table 8.17.

img_id	***chunknumber***	***type***	**Expected return value**	**Verification**
2	0	0	NO_SUCH_IMAGE	√
3	0	1	DONE	√
4	0	1	NO_SUCH_IMAGE	√
4	0	0	DONE	√
3	300	3	NO_SUCH_CHUNK	√
3	10	3	DONE	√
0	0	2	DONE	√
0	13	5	DONE	√

Table 8.17: *Shows the arguments used when cam_send_img was called, the expected return values were received.*

Furthermore, the size of the chunks did not exceed 10 kB, but the data downloaded could not be used, when a large amount of data were downloaded.

Conclusion

The module Send Image performs as expected on the microcontroller, but the PC software does not support of downloading large amount of data, thus it fails during the download procedure. This error is not corrected due to lack of time, but is expected to be corrected before examination.

8.3 Integration Test

To verify that the hardware, software and programmable logic are able to co-operate properly, a system integration test is performed which uses several software modules and hardware blocks. The functionality of the used modules and blocks have already been verified in previous tests and they should thereby be eliminated as a source of error if the test fails.

8.3.1 Testing Read-out Functionality

The test mode of the image sensor will be used to test the Read-out functionality of the camera system. This test will make sure the microcomputer, including its external control logic, is able to perform DMA control for the image sensor. The test make use of several modules to interface with the image sensor and hardware blocks for memory control.

Test Mode of the Image Sensor

The image sensor features a test mode which ignores the actual pixels captured and instead output known test values. By using this test mode it is possible to verify the data transfer from the image sensor works properly. The received data from the image sensor can be compared to known value, instead of estimating if an image matches a subject. This method ensures that possible errors in the configuration of the image sensor and adjustments of the lens system can be held separate from possible errors in the data interfaces.

The test data from the image sensor will be referred to as a test image as it is arranged just as an image. The test image contains the known test value in each even column, and the inverse in each odd columns. A column in the image sensor is an RG or GB pair as can be seen on Figure 8.6.

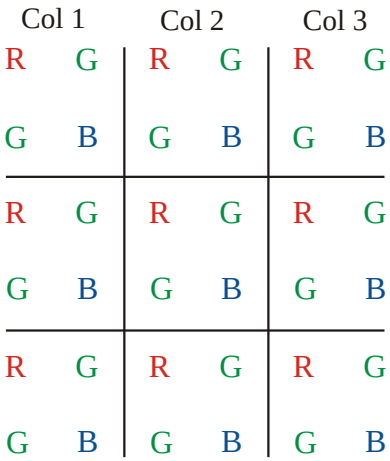


Figure 8.6: *Format of test image produced by the image sensor. All the pixels in col 1 will contain the value set in 0x32, and col 2 will contain the inverse. Col 3 will also contain the value and col 4 the inverse and so forth.*

Test Procedure for Testing Read-out

The camera system must be powered up and communication with HSNTest must be started.

- Enable test mode by setting bit 6 in register 0x07 using cam_setup(**data*, *length*).
- Transfer a test value with one bit set to register 0x32 using cam_setup(**data*, *length*).
- Call initialize_image_sensor.
- Jump to SRAM software.

- Analyze the transferred data in RAM3 and verify that the test image contains correct data. The first pair of bytes must contain the test value, the second pair must contain the inverse test value, the third pair should again contain the test value and so forth.
- The test is repeated seven times with a new test value with a different bit set. Thereby all eight data bus connections can be verified.

Conclusion of Read-out Test

For all eight tests the correct test image was produced. This means that the microcomputer is able to interface properly with the image sensor and the read-out functionality works as specified.

8.4 Acceptance Test

The purpose of this section is to perform the tests specified in Section 4.1, page 44. Some of the tests have already been accomplished through Section 8.2 meaning that only the untested requirements is tested in this section.

8.4.1 Results of Testing §SAT9

The camera system has to use either 3.3 V (±0.2 V), 5 V (±0.2 V) or both voltage levels to follow **§SAT9**. This demand is met, because all components operate on 3.3 V or 5 V except the FPGA which requires 2.5 V. A buck converter converting 3.3 V to 2.5 V is placed on the PCB to supply internal core logic of the FPGA, which means that only 3.3 V and 5 V is necessary to operate the camera system and **§SAT9** is therefore met.

8.4.2 Results of Testing §SAT10

The power budget of the camera system is set at 500 mW in active mode. To test this requirement the power consumption was read from the power supply, HM7042 AAU nr. 33880, while capturing an image and during a write cycle to FLASH, because it is observed that in these two situations the power consumption is largest. The results is shown in Table 8.18.

Supply voltage	Supply current	Power
Capture Image		
2.5 V	2 mA	5 mW
3.3 V	90 mA	297 mW
5 V	70 mA	350 mW
Total		652 mW
Writing to FLASH		
2.5 V	2 mA	5 mW
3.3 V	110 mA	330 mW
5 V	10 mA	50 mW
Total		385 mW

Table 8.18: *Power consumption during Capture Image and Writing to FLASH.*

Table 8.18 shows that **§SAT10** is not met, because the camera system uses 652 mW while capturing an image. The duration of this consumption is a few seconds and is primarily used by the image sensor on the 5 V supply.

8.4.3 Results of Testing §SAT11

The camera system must be able to communicate with the OBC by a HSN protocol via a CAN 2.0B bus. This requirement is tested by sending commands from HSNTest running on a PC via an USB to CAN converter. The requirement is met, because the camera system can receive and send HSN packets.

8.4.4 Results of Testing §FUNC3

The images captured by the camera system must be color images. To meet this requirement an image sensor using an RGB Bayer pattern is used [Micron, 2005, p. 1]. Thereby, the image is interpolated and the image compression is designed to compress color images as explained in Section 7.7, page 98.

8.4.5 Results of Testing §FUNC8

The non-volatile memory must be able to hold 5 compressed images and 5 thumbnails. This requirement is met, because the layout of the memory in FLASH is as shown on Figure 6.4 in Section 6.4.1, page 80. In this memory layout space is reserved to 5 compressed images and 5 thumbnails.

8.4.6 Results of Testing §FUNC9

It must be possible to change the following camera settings, gain (light sensitivity) and duration of exposure. This requirement is met, because all the settings including gain and duration of exposure can be set on the chosen image sensor [Micron, 2005, p. 12]. The settings can be changed by writing new settings to the image sensor using the serial protocol described in Section 7.2.1, page 92.

8.4.7 Conclusion of Acceptance Tests

All tested requirements presented in this section except **§SAT10** are met. This means that some redesigns are necessary to minimize the power consumption of the camera system before all requirements are met.

9 Conclusion

Even though no images with an actual subject were retrieved from the camera system, it has been shown that it is possible to design and make a prototype of a digital camera system for a pico satellite within the time frame of a semester.

The project was initiated as a feasibility study with the purpose of designing and implementing a microcomputer system to interface a camera. The prototype is designed in preparation for using the system in a pico satellite; more specifically for AAUSAT-III. It was researched which challenges that arises when designing electronics for use in space. The physical dimensions, mass, and power consumptions must be small due to the limited resources and space in a pico satellite. Besides, the camera system must be able to operate in a large temperature range, operate in vacuum, and be capable of withstanding a larger amount of radiation than on the Earth. Being a subsystem on a satellite, the camera system must also be reliable and be able of communicating with the on-board computer of the satellite.

As the mission for AAUSAT-III is not yet specified, it was necessary to define the interface and the requirements from former satellite projects. Here, focus was directed toward the AAU space program and the requirements from the two existing projects were investigated to form the expected requirements for the developed camera system prototype. As the AAUSAT-II is developed upon the experiences from the AAU Cubesat, the design for AAUSAT-II subsystems were found to be the most feasible of the two for using as a reference when designing the subsystem for AAUSAT-III. As so, the prototype developed closely resembles the AAUSAT-II subsystems and is in the current prototype possible to use with a second generation of the AAUSAT-II.

The design that is implemented allows for a flexible optical design, as the camera system is developed in a form where it can be mounted easily within a satellite due to the division in two PCBs; a larger microcomputer system and a smaller image sensor board. Also the current system features no restraints on the settings of the image sensor, so the image sensor and the optical system can be configured to match each other.

The components used for the prototype is chosen with care to comply with the demands to vacuum, radiation, power consumption, and size. This is done with the assumption that an early awareness to these requirements will mean less development time from prototype to flight model.

The prototype has debugging facilities implemented that is not designed to be suitable for use in space, but apart from the debugging facilities, space graded requirements from ESA have been followed for the PCB layout. This features like hidden signal layers, glass polyimide laminate, and almost only trough-going vias on the PCB. Besides the ESA requirements V_{CC} and GND planes are included in the PCB to minimize noise. The sheer size of the prototype clearly indicates that it is possible to create a PCB that can be fitted inside a pico satellite.

The microcomputer system is based on a microcontroller based on the ARM7 core and both FLASH and RAM are available for respectively long and short term storage. There is enough memory installed to handle a 3 Mpixel raw image as long as it does not have to be interpolated. In the present configuration of the camera system, only an image of 1.5 Mpixel can be handled. In this configuration the memory can contain a raw image and five compressed images and five thumbnails. The DMA used for capturing an image is implemented in a FPGA; allows for flexibility and more advanced DMA features.

It has been shown that the hardware design is fully functional, and that the DMA implementation in a FPGA is fully capable of retrieving the 8 most significant bits of the data that the image sensor produces when capturing an image.

Control of the image sensor is implemented in embedded software running on a processor currently used on AAUSAT-II, and a low level software has been implemented to control the DMA, utilizing the internal RAM of the ARM, when addressing external memory is not possible.

Within the six days used to implement the system, two prototypes were manufactured but the first one of the two was never capable of booting the operating system used. However, it was used to test the functionality of the FPGA, ensuring that the VHDL software was synthesized correctly before any attempt was made to insert it as DMA controller in the second prototype.

No adjustments where made to the hardware design during implementation, but the system was improved as the test from the FPGA showed continuously that the code was synthesized and therefore could be made more advanced than the original design. This knowledge gained was used to implement an extra safety feature to ensure that no bus conflicts could emerge on the data bus even if the ARM could contain programming glitches that would make it try to write data to an external address during DMA progress.

To control the hardware of the camera system an embedded software is implemented running on the ARM microcontroller. The software is based on the embedded operating system eCos. The advantages gained from using a generic operating system is obvious, as it provides dynamic memory allocation, protection of variables between objects running multiple processes, and scheduling between the processes. The application layer that is installed on the operating system handles communication with the on-board computer hardware, including e.g. the FLASH. It accesses the FLASH and stores data and reports all errors that occur. Especially the error handling is important for systems running on a satellite, as it is only the data produced by the systems them selves that can be used for diagnosis of the system.

When communicating with the system from an autonomous system as CDH, it will ease the design of the autonomous system if it can access its subsystems at all times. To help the CDH programmers it has been decided to implement a command buffer that will hold all incoming commands till they are ready for execution. When commands are completed or errors occur, replies are made that is identifiable and can be put in a log for later analysis. Also housekeeping data can be fetched that contains updated information about temperature and memory status of the system.

As the mass of the flight PCB and optics are not yet known, it is not possible to finally determine if the camera system can be placed within the mass budgets of AAUSAT-III. However, it is likely as the system is based on relative few and small components. The power budget that was specified is almost achieved in the prototype developed, and it should be possible to lower the power consumption before launch, as this has not been the primary concern in the development phase. Also the final design cannot be completed before the mission for the satellite is defined.

The design implemented is based on components that were not within the textbooks or lectures of the semester. However, by being watchful when designing hardware and software in parallel, using iterations and documentation to improve the design, it was possible to design the hardware, the programmable logic, and the software for a digital camera to the degree that made it possible to assemble an electrically fully functional prototype within only six days without altering the design. Even so, there was not enough time left before deadline to adjust the lens and the settings for the image sensor to capture an image of an external subject.

The prototype PCB arrived only nine days before the deadline, and after six days of hardware implementation and FPGA optimizing there was no time for optimizing the software, even so only a very few functionalities were not fully adapted to the hardware. The software implemented goes beyond the textbooks used in the semester, as it not only controls and interacts with the hardware, but also introduces object oriented software. The software design implemented is based on embedded software, and utilizes three communication standards to operate the hardware. One of these is implemented as a low level protocol, handling the interface with the image sensor and temperature sensor on the image sensor board. This follows the guideline of the semester that specifies that a serial communication must be implemented on the system.

The semester also yields that a communication must be achieved with another computer. To be able to communicate with the system from a PC, a win32 software was modified with a new GUI to handle the camera system. Communication with the prototype was successfully tested using this software. The debug interface using the RS-232 standard is accessible from any hyper terminal, such as minicom for Linux or HyperTerminal for Windows.

Before the system is capable of capturing an image with a recognizable subject, the lens must be adjusted to fit the image sensor. As the prototype was only completed very shortly before the turn in of the report, it has not been tested yet how well the system performs when capturing recognizable subjects.

The lens adjustment is expected to be time consuming as Devitech ApS supplied a lens that not only can be adjusted for zoom and focus, but also can be mounted in two different configurations (C and CS mount); changing their focal point drastically. It has not yet been confirmed which configuration is optimal but when this is placed it is expected that an image is captured without complications.

10 Perspectives

The project features many different areas of development and in many of them a compromise between available time and developed ideas had to be committed. As a result some tasks remain that should be completed before the system is ready for flight and therefore should be implemented after the project time has expired.

One of the remaining tasks is to complete the test of the camera system according to the Test specifications to make sure it performs as specified. It is especially important when a circuit is to be launched into space, as the repair facilities in space are limited to software updating.

Before the camera system is ready to be finally tested a few adjustments still needs to be done.

This section describes the adjustments in two parts, each containing different priorities of tasks. In Section 10.1 the primary adjustments are described to the extend they are already known. In Section 10.2 secondary possibilities containing improvements that were found feasible are described.

10.1 Adjustments Before Flight

In this section the known adjustments, which need to be corrected before flight are described. These should be implemented before launching a satellite with the camera system on board.

Smaller PCB

The PCB is designed for a variety of debug facilities. This includes JTAG, RS-232 and a wealth of jumpers allowing for multiple hardware configurations. Vital improvements of the PCB is already mentioned in the report in Section 5.3.2, page 62. The prototype PCB even though it is small, is not designed to be fitted in a pico satellite. Before this is attempted, the debugging facilities should be removed from the PCB and the components should be moved closer to each other so that it can fit inside AAUSAT-III. However, this can not be done till the AAUSAT-III budgets are defined for size and mass of individual subsystems.

Software Update Facility

As the RS-232 and JTAG are not expected to be used in the flight models, it should be possible to upload and activate new software images over the CAN. Should HSN be chosen for AAUSAT-III, there can easily be implemented a HSN extension of the existing software upload system. The system is in the prototype capable of handling software uploads over the RS-232 and the functions implemented for this should be sufficient to handle the writing of a software image to FLASH and the following reboot to this.

Optimizing Algorithms for Image Manipulation

The current speed of the implemented image compression algorithms is still not optimal. It was not the main goal of this project to implement a fully optimized compression algorithm but simply to create one that was possible to use within the limits of the camera system. The one used currently takes several minutes, and thus, as a result uses relative too much power. This is an obvious target for improvement. It is possible, if the one currently used cannot be optimized enough, that another algorithm could be implemented for this task.

Ensure a Proper Lens System

The lens used for the prototype is provided by Devitech ApS and is in no way designed for use in space. A lens system must be found and installed on the system. The image sensor board provided by Devitech ApS is designed to use a C or CS mounted lens.

Update the BlackEye Board

The project group has collected the components and a ready to solder PCB from Devitech ApS. This allows for a class three soldering even though the buffer is a BGA component. However, the PCB is a FR4 commercial PCB, and should if possible be upgraded to Space Graded Glass Polyimier. Devitech ApS has provided schematics and layout files making it possible to change the PCB laminate type. When this is to be done anyway, it should be considered to remove the use of cables between the board and the microcomputer; there are two obvious ways to do this:

1. Including the Devitech BlackEye board into the layout of the camera system. This would make it possible to have the entire subsystem in a single PCB.

2. If it should be required by MECH or the lens design to have the image sensor board separated from the microcomputer as is the case in the current design, the connections between the PCBs should be made using flex print. Should this be chosen, the deal with GPV Printca A/S that was settled also defines a very attractive price for flex print.

Temperature Measurement of the Microcomputer

The image sensor board from Devitech ApS is designed with a temperature sensor, which is accessible through a serial interface using I^2C for communicating. This is monitored by the software and used for determining if the image sensor is allowed to be turned on. Housekeeping data is also retrieved from this temperature sensor and it is possible to collect this using the HSN. However, before the camera is send into space, temperatures of the entire system should be monitored.

10.2 Improvements of the Design

In this section suggestions are given on how to improve the design, simply by adding new features to the existing.

Multiple Resolutions

As the image sensor can produce up to 3 Mpixel images it should be considered to make all 3 Mpixels available for the user. The existing hardware can be used to capture a raw 3 Mpixel image, and hold it temporarily in RAM. However, the current interpolation algorithm that is implemented can not interpolate the 3 Mpixel image on the hardware, as it tripples the raw data when creating the RGB channels. Due to the limited amount of memory it would be needed to drastically change the memory use of the interpolation and compression methods.

A suggestion could be to interpolate and compress small areas and save them in FLASH before moving on in the raw data. There is already sufficient memory in FLASH where the 1.5 Mpixel raw data is currently being saved to instead store a compressed 3 Mpixel image, without changing the compression ratio. As a result this would mean that transferring the image will take longer time than a 1.5 Mpixel.

Devitech ApS Interpolation Algorithm

While settling the agreement with Devitech ApS for the supply of an image sensor, a NDA was agreed upon. Protected by this agreement was a promise of delivering an algorithm for interpolating Bayer formats. The algorithm was not handed over within the time frames of the project, but the agreement with Devitech ApS still stands. As so better quality on the images produced could be achieved, if the interpolation algorithm is implemented.

Adjustment toward Mission

The current prototype is developed from a concept of photographing the Earth. However, as it was mentioned in the introduction, there exists other possibilities for the use of the camera system. The system can be easily adapated to other missions, by designing a lens system. The image sensor chosen does not have an IR filter, and a set of filters can be applied on the lens to control the

sensitivity to colors. Dependent on the mission the resolution and the compression ratio can be adapted, as smaller images could allow for more than five images.

Hamming Coding

As was explained in the analysis, techniques exist to encode data so it is more resistant to bit flips. Hamming coding could be a feasible way to improve the reliability of data. However, as it was also seen in the analysis, the shielding of the satellite should be sufficient to prevent most of the damage caused by radiation. So this is not implemented in the camera system in this version, it is suggested that this is implemented at least on the default settings and software.

Bibliography

[03gr731, 2003] 03gr731 (2003). Designing prototyping and testing of a flexible On Board Computer platform for pico-satellites (Worksheet).

[04gr720, 2004] 04gr720 (2004). AAUSAT-II CDH Project Report.

[AAU, 2002] AAU (2002). Attitude Determination for AAU CubeSat. http://www.cubesat.auc.dk/dokumenter/ADC-report.pdf, 02/11-2006.

[AAUSAT-II, 2004] AAUSAT-II (2004). AAUSAT-II Power Budget. AAUSAT-II CVS, 02/08-2006.

[AAUSAT-II, 2005a] AAUSAT-II (2005a). AAUSAT-II About. http://aausatii.aau.dk/wiki/index.php/About, 02/08-2006.

[AAUSAT-II, 2005b] AAUSAT-II (2005b). AAUSAT-II Datarate Budget. AAUSAT-II CVS, 02/08-2006.

[AAUSAT-II, 2005c] AAUSAT-II (2005c). AAUSAT-II EPS Project Report. AAUSAT-II CVS, 02/08-2006.

[AAUSAT-II, 2005d] AAUSAT-II (2005d). AAUSAT-II Mass Budget. AAUSAT-II CVS.

[AAUSAT-II, 2005e] AAUSAT-II (2005e). AAUSAT-II On-board Computer System. http://aausatii.aau.dk/wiki/index.php/On-board_Computer_System, 03/04-2006.

[AAUSAT-II, 2006] AAUSAT-II (2006). AAUSAT-II HSN Protocol Documentation. http://aausatii.aau.dk/wiki/index.php/HSN_protocol_documentation, 02/20-2006.

[AMD, 2005] AMD (2005). Am29LV320MT/B Data Sheet. http://www.amd.com.cn/CHCN/assets/content_type/white_papers_and_tech_docs/26518c1.pdf, 03/04-2006.

[ASE, 2000] ASE, T. A. f. S. E. (2000). Plug in the Sun. ISBN: 0-86357-314-2 http://www.sycd.co.uk/is_there_life/pdf/engage/pits.pdf, 05/27-2006.

[Atmel, 2003] Atmel (2003). Aerospace Products Radiation Policy. http://www.atmel.com/dyn/resources/prod_documents/doc4170.pdf, 03/01-2006.

[Atmel, 2004] Atmel (2004). Drop-In/Stand-alone Programming Circuits for AT17 Series Configurators with Atmel and Xilinx® FPGAs. http://www.atmel.com/dyn/resources/prod_documents/doc3032.pdf, 05/22-2006.

[Atmel, 2005] Atmel (2005). AT91 ARM®Thumb®-based Microcontroller - AT91SAM7A1. http://www.atmel.com/dyn/resources/prod_documents/doc6048.pdf, 02/20-2006.

[Atmel, 2006] Atmel (2006). FPGA Configuration EEPROM Memory - AT17LV512. http://www.atmel.com/dyn/resources/prod_documents/doc2321.pdf, 04/02-2006.

[Bak-Jensen, 2006] Bak-Jensen, B. (2006). Elektromagnetisme 4.sem mm1. http://kom.aau.dk/~heb/kurser/felt-06/eloh01v3.pdf, 04/26-2006.

[Benamati, 2003] Benamati, B. (2003). Accumulation-mode readouts benefit interline CCD users. http://wwwuk.kodak.com/global/plugins/acrobat/en/digital/ccd/papersArticles/Accumulationmode.pdf, 03/03-2006.

[Benedetto, 1998] Benedetto, J. M. (1998). ECONOMY-CLASS ION-DEFYING ICs IN ORBIT. http://ams.aeroflex.com/ProductFiles/Articles/IEEERadHard.pdf, 03/01-2006.

[Bhanderi, 2006] Bhanderi, D. D. (2006). Satellite Development. http://cpk.auc.dk/education/SSU-2006/mm5/E4-6-mm5.pdf, 03/11-2006.

[Bierring-Sørensen et al., 2002] Bierring-Sørensen, S., Hansen, F. O., Klim, S., and Madsen, P. T. (2002). Håndbog i Struktureret Program Udvikling. Ingeniøren—bøger, 1. edition. ISBN: 87-571-1046-8.

[Bosch, 1991] Bosch, R. (1991). Bosch CAN v2. Specifications. http://www.semiconductors.bosch.de/pdf/can2spec.pdf, 02/23-2006.

[Bosch, 1999] Bosch, R. (1999). The Configuration of the CAN Bit Timing. http://www.semiconductors.bosch.de/pdf/CiA99Paper.pdf, 03/01/2006.

[Burd, 1995] Burd, T. (1995). Power Dissipation in CMOS ICs. http://bwrc.eecs.berkeley.edu/Publications/theses/low.power.CMOS.library.MS/power.2.html, 03/06-2006.

[Chang, 1997] Chang, K. (1997). Digital Design and Modeling with VHDL and Synthesis. IEEE Computer Society Press. ISBN: 0-8186-7716-3.

[Chang, 2002] Chang, Y. K. (2002). Vacuum Environment. galaxy.yonsei.ac.kr/board/act.php?o%5Bat%5D=dn&dn%5Btb%5D=pds&dn%5Bcd%5D=20021017160958&dn%5Bname%5D=vacuum.pdf, 03/12-2006.

[Christiansen, 2004] Christiansen, J. (2004). Radiation Hardness Assurance. http://lhcb-elec.web.cern.ch/lhcb-elec/html/radiation_hardness.htm, 03/01-2006.

[CiA, 1994] CiA (1994). CAN Physical Layer for Industrial Applications. http://www.phytec.com/manuals/ds102.pdf, 02/24-2006.

[Coorporation, 2003] Coorporation, N. S. (2003). BGA Ball Grid Array. http://www.national.com/an/AN/AN-1126.pdf, 05/19-2006.

[Corporation, 2006] Corporation, A. (2006). PLL Basics. http://www.altera.com/support/devices/pll_clock/basics/pll-basics.html, 04/22-2006.

[Cypress, 2005] Cypress (2005). CY62167DV30. http://www.cypress.com/portal/server.pt/gateway/PTARGS_0_2_1524_209_259_43/http%3B/sjapp20%3B7001/publishedcontent/publish/design_resources/datasheets/contents/cy62167dv30_5.pdf, 03/04-2006.

[DALSA, 2005] DALSA (2005). CCD vs. CMOS. http://www.dalsa.com/markets/ccd_vs_cmos.asp, 02/08-2006.

[Davis, 1998] Davis, L. (1998). Encoding Dictionary, terms, and definitions. http://www.interfacebus.com/Definitions.html, 03/06-2006.

[Devitech, 2004] Devitech (2004). Devitech ApS Website. http://www.devitech.dk, 04/10-2006.

[Ebert, 2006] Ebert, H. (2006). EMC MM2. http://kom.aau.dk/~heb/kurser/emc-06/eoh02.pdf, 04/26-2006.

[EMT, 2006] EMT (2006). Electronics Manufacture and Test Glossary. http://www.emtonthenet.net/glossary/fr4laminate.html, 05/22-2006.

[Epson, 1998] Epson (1998). SMD HIGH-FREQUENCY CRYSTAL UNIT - MA-505/MA-506. http://www.comtec-crystals.com/datasheets/ma505_506/ma505_6.pdf, 05/11-2006.

[ES2, 1993] ES2, E. S. S. (1993). Technology & Services. European Silicon Structures. Appendix 1: ES2 Asic Design Guidelines.

[ESA, 2006] ESA (2006). ESA Facts and Figures. http://www.esa.int/esaCP/GGG4SXG3AEC_index_0.html, 05/09-2006.

[Faccio, 2000] Faccio, F. (2000). COTS for the LHC radiation environment: the rules of the game. http://alicedcs.web.cern.ch/AliceDCS/ElectrCoord/Documents/FF_COTS_paper.pdf, 03/07-2006.

[Filippi, 2006] Filippi, P. (2006). AT91 Support Group. 05/22-2006.

[Forslund et al., 1999] Forslund, K. E., Fregil, F., and Moth, K. (1999). Danish Small Satellite Programme Study of Electronic Components for Small Satellites Final Report. www.dsri.dk/roemer/pub/Documents/ECSS_1F.PDF, 03/10-2006.

[Freescale, 2006] Freescale (2006). MC68000 Product Summary Page. http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MC68000, 04/30-2006.

[Golson, 1993] Golson, S. (1993). One-hot state machine design for FPGAs. http://www.trilobyte.com/pdf/golson_pldcon93.pdf, 05/16-2006.

[Group, 1998] Group, I. J. (1998). JPEG image compression. http://www.ijg.org/, 05/06-2006.

[Hansen, 2001] Hansen, F. (2001). DTU Satellite Systems and Design Course. http://www.cubesat.auc.dk/documents/Space_Environment.pdf, 03/01-2006.

[Henrique S. Malvar and Cutler, 2004] Henrique S. Malvar, L.-w. H. and Cutler, R. (2004). High quality linear interpolation for demosaicing of Bayer-patterned color images. http://research.microsoft.com/~rcutler/pub/Demosaicing_ICASSP04.pdf, 02/22-2006.

[Holbert, 2006] Holbert, D. (2006). Single Event Effects. http://www.eas.asu.edu/~holbert/eee460/see.html, 02/27-2006.

[Instruments, 2005] Instruments, T. (2005). TMP100 TMP101. http://www.ortodoxism.ro/datasheets2/6/0rshokqip62u8lgwokofk1ph6ffy.pdf, 05/18-2006.

[Janesick, 2002] Janesick, J. (2002). Dueling Detectors. http://www.dalsa.com/shared/content/OE_Magazine_Dueling_Detectors_Janesick.pdf, 02/08-2006.

[Kodak, 2003] Kodak (2003). Shutter Operations for CCD and CMOS Image Sensors. http://www.kodak.com/global/plugins/acrobat/en/digital/ccd/applicationNotes/ShutterOperations.pdf, 05/27-2006.

[Kramer, 2006] Kramer, H. J. (2006). Status on the First Multiple CubeSat Launch. http://directory.eoportal.org/pres_CubeSatLaunch1.html, 02/20-2006.

[Larsen, 2006a] Larsen, L. B. (2006a). E4-6: Struktureret Systemudvikling. Struktureret Systemudvikling mm3, http://cpk.auc.dk/education/SSU-2006/mm1/E4-6-mm3.pdf, 02/22-2006.

[Larsen, 2006b] Larsen, L. B. (2006b). E4-6: Struktureret Systemudvikling - mm1. Struktureret Systemudvikling mm1, http://cpk.auc.dk/education/SSU-2006/mm1/E4-6-mm1.pdf, 02/22-2006.

[Laursen, 2006] Laursen, K. K. (2006). HSN protocol documentation - AAUSAT-II. http://aausatii.aau.dk/wiki/index.php/HSN_protocol_documentation, 02/26-2006.

[Lee and Tepfenhart, 2001] Lee, R. C. and Tepfenhart, W. M. (2001). UML and C++, A Practical Guide To Object-Oriented Developement. Prentice Hall. ISBN: 0-13-029040-8.

[Litwiller, 2001] Litwiller, D. (2001). CCD vs. CMOS: Facts and Fiction. http://www.dalsa.com/shared/content/Photonics_Spectra_CCDvsCMOS_Litwiller.pdf, 02/08-2006.

[Massa, 2002] Massa, A. J. (2002). Embedded Software Development with eCos. Prentice Hall, 1. edition. ISBN: 0130354732.

[Maxim, 1999] Maxim (1999). MAX3222/MAX3232/MAX3237/MAX3241. http://pdfserv.maxim-ic.com/en/ds/MAX3222-MAX3241.pdf, 03/22-2006.

[Micron, 2005] Micron (2005). MT9T001 - 1/2-Inch 3-Megapixel Digital Image Sensor. http://download.micron.com/pdf/datasheets/imaging/MT9T001_3100_DS.pdf, 04/27-2006.

[Micron, 2006] Micron (2006). Lux and Light - Illumination, Exposure, and Sensitivity. http://www.micron.com/products/imaging/technology/lux.html, 02/11-2006.

[muRata, 1998] muRata (1998). Differential and Common Mode Noise. http://www.murata.com/emc/knowhow/pdfs/te04ea-1/26to28e.pdf, 05/08-2006.

[OMG, 2004] OMG (2004). Unified Modeling Language: Superstructure version 2.0 formal/05-07-04. http://www.omg.org/docs/formal/05-07-04.pdf, 04/22-2006.

[Ott, 1988] Ott, H. W. (1988). Noise Reduction Techniques in Electronic Systems. John Wiley & Sons, 2. edition. ISBN: 0-471-85068-3.

[Poivey et al., 2002] Poivey, C., Gee, G., Barth, J., LaBel, K., and Safren, H. (2002). Draft. http://radhome.gsfc.nasa.gov/radhome/papers/2002_SSR.pdf, 03/01-2006.

[Popescu and Farid, 2005] Popescu, A. C. and Farid, H. (2005). Exposing Digital Forgeries in Color Filter Array Interpolated Images. http://www.cs.dartmouth.edu/~popescu/research/papers/pdf_files/sp05.pdf, 02/27-2006.

[Ramanath et al., 2002] Ramanath, R., Snyder, W. E., Bilbro, G., and III, W. A. S. (2002). Demosaicking methodes for Bayer color arrays. http://home.comcast.net/~rramanath/Research/demosaicking-JEI-02.pdf, 03/01-2006.

[Rosenberg, 2001] Rosenberg, C. (2001). The Lenna Story. http://www.lenna.org, 05/06-2006.

[Ryer., 1997] Ryer., I. L. A. D. (1997). The Light Measurement Handbook. http://files.intl-light.com/handbook.pdf, 02/11-2006.

[SEGGER, 2005] SEGGER (2005). J-Link ARM: General info. http://www.segger.com/jlink.html, 03/24-2006.

[Semiconductors, 2003] Semiconductors, N. (2003). LM2619. http://www.national.com/ds.cgi/LM/LM2619.pdf, 04/10-2006.

[Serway and Jewett, 2004] Serway and Jewett (2004). Physics for Scientists and Engineers. David Harris, 6. edition. ISBN: 0-534-40949.

[Sloss et al., 2004] Sloss, A. N., Symes, D., and Wright, C. (2004). ARM System Developer's Guide. Morgan Kaufmann. ISBN: 1- 55860-874-5.

[Sondrup, 1995] Sondrup, P. B. (1995). EMC designregler. http://kom.aau.dk/~heb/kurser/emc-06/emcnota2.pdf, 04/30-2006.

[Standard, 1998] Standard, I. (1998). ISO 12232 - Photography - Electronic still-picture cameras - Determination of ISO speed. Dansk Standard, 1. edition. Reference number: ISO 12232:1998(E).

[Tanenbaum, 1999] Tanenbaum, A. S. (1999). Structured Computer Organisation. Prentice-Hall International, 4. edition. ISBN: 0-13-020435-8.

[Tanenbaum, 2006] Tanenbaum, A. S. (2006). Structured Computer Organisation. Pearson Prentice Hall, 5. edition. ISBN: 0-13-148521-0.

[Titus, 2001] Titus, H. (2001). Sensors.
`http://www.kodak.com/global/plugins/acrobat/en/digital/ccd/papersArticles/sensorsCaptureAttention.pdf`, 03/03-2006.

[University, 2004a] University, C. P. S. (2004a). CubeSat Design Specification.
`http://littonlab.atl.calpoly.edu/media/Documents/Developers/CDS%20R9.pdf`, 02/18-2006.

[University, 2004b] University, C. P. S. (2004b). DNEPR Safety Compliance Requirements.
`http://littonlab.atl.calpoly.edu/media/Documents/Developers/compliance_dnepr_lv.pdf`, 02/23-2006.

[USNA, 2001] USNA (2001). Basic procedures for Cyclic, Hamming, Binary Reed-Muller, BCH, and Golay codes.
`http://web.usna.navy.mil/~wdj/codes.html`, 03/19-2006.

[van Bilsen, 2004a] van Bilsen, E. (2004a). AIC.
`http://www.bilsen.com/aic/index.htm`, 03/13-2006.

[van Bilsen, 2004b] van Bilsen, E. (2004b). AIC - Block Prediction.
`http://www.bilsen.com/aic/blockprediction.htm`, 03/13-2006.

[van Bilsen, 2004c] van Bilsen, E. (2004c). AIC - Color Conversion.
`http://www.bilsen.com/aic/colorconversion.htm`, 03/13-2006.

[van Bilsen, 2004d] van Bilsen, E. (2004d). AIC - Results.
`http://www.bilsen.com/aic/results.htm`, 03/13-2006.

[Wagner, 2002] Wagner, N. R. (2002). The Hamming Code for Error Correction.
`http://www.cs.utsa.edu/~wagner/laws/hamming.html`, 03/14-2006.

[Wakerly, 2000] Wakerly, J. F. (2000). Digital Design - Principles and Practices. Pearson Prentice Hall, 3. updated edition. ISBN: 0-13-089896-1.

[Wikipedia, 2006a] Wikipedia (2006a). Application Specific Integrated Circuit.
`http://en.wikipedia.org/wiki/Application-specific_integrated_circuit`, 05/15-2006.

[Wikipedia, 2006b] Wikipedia (2006b). Bayer filter.
`http://en.wikipedia.org/wiki/Bayer_filter`, 02/11-2006.

[Wikipedia, 2006c] Wikipedia (2006c). Charge Coupled Device.
`http://en.wikipedia.org/wiki/Charge-coupled_device`, 05/19-2006.

[Wikipedia, 2006d] Wikipedia, T. F. E. (2006d). f-number.
`http://en.wikipedia.org/wiki/F-number`, 05/21-2006.

[Wikipedia, 2006e] Wikipedia, T. F. E. (2006e). Field-programmable gate array.
`http://en.wikipedia.org/wiki/Fpga`, 05/01-2006.

[Wikipedia, 2006f] Wikipedia, T. F. E. (2006f). Joint Test Action Group.
`http://en.wikipedia.org/wiki/JTAG`, 05/19-2006.

[Wikipedia, 2006g] Wikipedia, T. F. E. (2006g). Programmable Logic Device.
`http://en.wikipedia.org/wiki/Programmable_logic_device`, 05/22-2006.

[Wikipedia, 2006h] Wikipedia, T. F. E. (2006h). YCbCr.
`http://en.wikipedia.org/w/index.php?title=YCbCr&oldid=42661460`, 03/12-2006.

[XILINX, 2001] XILINX (2001). Power-On Requirements for the Spartan-II and Spartan-IIE Families.
`http://www.xilinx.com/bvdocs/appnotes/xapp450.pdf`, 04/29-2006.

[XILINX, 2004] XILINX (2004). Spartan-II 2.5V FPGA Family: Complete Data Sheet. `http://direct.xilinx.com/bvdocs/publications/ds001.pdf`, 04/08-2006.

Appendix A

A.1 Methods Used in the Project

The process of designing and prototyping a microcomputer system like the one for the camera system for AAUSAT-III requires a variety of tasks to be completed. To get as far as possible on a single semester, a set of project management tools were introduced. The tools were selected from different project development methods and were combined to form the method used for the project.

This appendix treats the method used for the project management. The method is combined from the suggestions made for iterative methods presented in the SSU lectures. After the method is generally described the implementation of the method is described briefly where it can be recognized in the report. Last in the appendix in Appendix A.1.3 the tools used for debug and development is briefly described.

A.1.1 Project Phases

From the SSU course a variety of tools were introduced, and three sets of documents were produced during the project in connection with the lectures. The three documents where all written in a form where they could be assimilated into the final report.

- An analysis document that documents the problems defined through the analysis and review of requirements.
- A design document containing the software design.
- A users manual for defining an easy use of the system.

The project was planned using a crossover model between SSU and UML. It is inspired from one of the lectures in SSU where different implementations of structured development models where presented [Larsen, 2006a]. The model utilizes iterations and describes the work in phases as illustrated on Figure A.1. The specific model is adapted to this projects by the project group, to compensate for the lack of external clients that standard models assume that there exist clients who sets some of the specifications; this is not the case for many of the student projects.

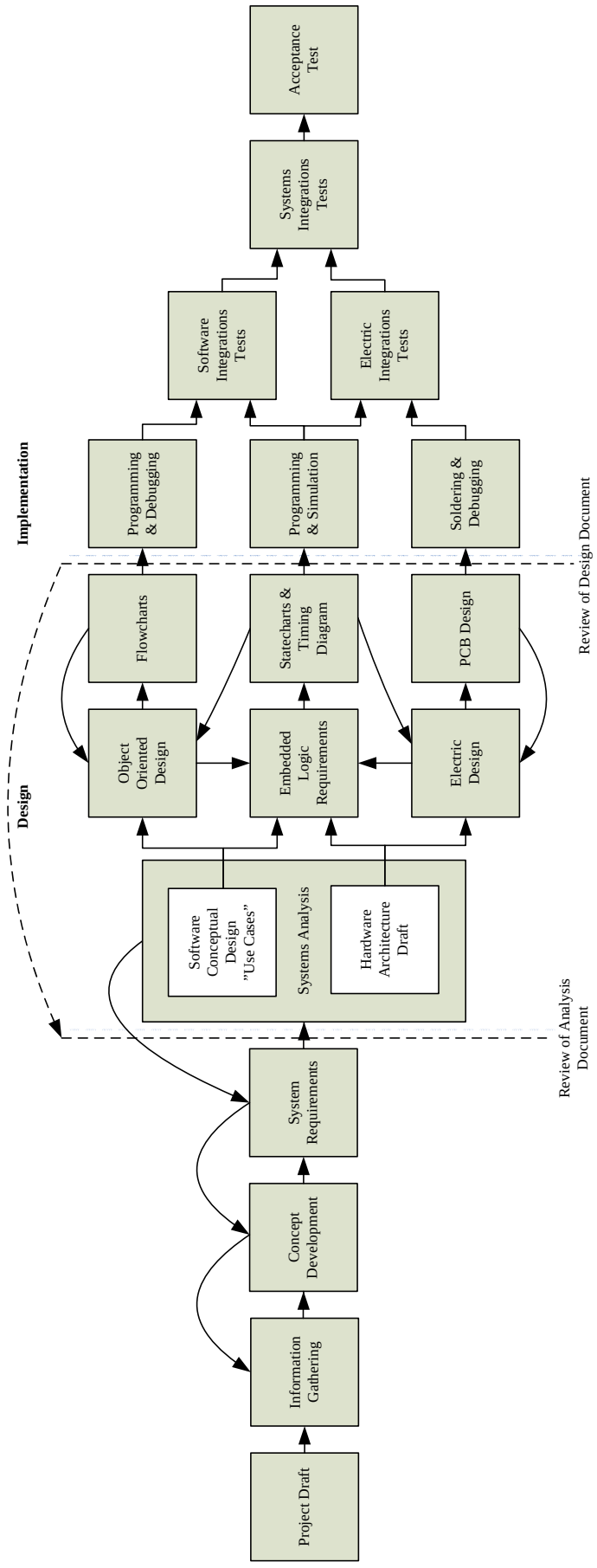


Figure A.1: *The model used for project management. It is evolved from [Larsen, 2006a, p. 35]. Backward pointing arrows indicate an iteration. Dashed vertical lines a deadline for review. The design phase includes a long jump iteration, this indicates redesigning is needed after the documentation is reviewed*

Phase One: Project Draft

The project proposal is developed into a project draft that more clearly specifies what is to be investigated before the requirement specifications can be outlined. In this phase brainstorming and interviews can provide some of the basic knowledge needed for this. The gained knowledge afterward serves as a plan for the areas that should be investigated before a design can be made.

Phase Two: Information Gathering

The research that the draft proposes is carried out to the necessary extend for setting up reasonable requirements. This phase is crucial to the further development of the project as it is the information gathered that will provide the base for the requirements and the conceptual project layout. In this phase books, experts, papers, and on-line material are gathered and analyzed for useful information. The information found is then summarized to get an overview of the knowledge gained by the members of the project group.

Phase Three: Concept Development

When enough information is gathered to get a feel of the possibilities in the project a basic concept for the project is laid out, aiding in the concept development SSU provides a toolbox that can be combined with the UML based "use cases". In combination they provide a valuable tool to identify the necessary functionalities of a system. The use cases combined with the summaries can be used for setting up all the requirements for the designed system.

Phase Four: System Requirements

In this phase the document with gathered information and the functionalities detected as necessary is used to specify the system requirements. The system requirement specifications gives a list of specifications that must be satisfied before the system is ready to use.

While specifying a requirement, a procedure for testing the requirement must also be conceived so the system can be verified for functionality and reliability after development. Tests help revealing if all requirements actually can be tested. Requirements that cannot be tested is not requirements.

This phase ends up in an analysis document that is reviewed by an external group. This gives a clue to detect possible misconceptions or erroneous conclusions.

Phase Five: System Analysis

In the system analysis the requirement specifications are used to narrow the selection of components to a selection that fits to the physical requirements of the system. Physical requirements are temperature, humidity, radiation, and the likes. The physical requirements can severely limit the choice of components, and the functionality required needs to be divided into software and hardware where it is possible to get an overview. The software is then viewed upon from an object oriented perspective and any obvious connections are made in the hardware layout should any exist.

Phase Six: Design Phase

As the design of the software is further specified into functions and algorithms, requirements emerge to the hardware concerning CPU power and memory layout. This hardware can be connected and calculated to a point where the bottle necks of the system can be identified. Then choices have to be taken to move about the software and hardware functionality if the bottle necks are causing troubles.

Functionality in hardware has become a more realistic choice for many applications, as ASICs have become yet more advanced through out the years. However, for some tasks that combines the software with the hardware interfaces it can be an advantage in the debugging phase to be able to change the relations between hardware and software interfacing. For this programmable logic can be introduced to give a flexible design, allowing for applications to use the speed of the hardware, and the flexibility of the software. Implementation of such components are described by the term co-designed hardware and software.

The design process is based on iterations. As the design progresses, the hardware and the software continuously increase the requirements to the co-designed components that is to connect the two areas. These components are during this process for each functionality implemented also setting stricter demands to the hardware and software design.

The method is build up from a model that assumes that the design is not necessarily fully functional from the first design attempt. However, after a few iterations of the design, it has reached a point where their interactions are outlined and each area can be further developed independently.

The design process is based upon the knowledge gained in the System Analysis. However, when the System Analysis show that prior known technology and experience is not sufficient to detect an obvious solution, new technology can be implemented. However, implementation of this in the design should always be done with great care as the lack of experience with the technology opens an extra margin for errors.

It is likely, especially when introducing new technologies, that it takes a few iterations to achieve a design that fulfill the requirements. As soon as a design exists that meets the requirements, the design documents for each of the design areas are combined to a unified design document, and this is then reviewed by the entire project group to identify possible lacks in the design. Designed modules can be tested and verified using evaluation boards or similar test boards.

Phase Seven: Implementation of Design

When the design document is approved the hardware design can be outlined to a PCB design, the software can be converted into code, and the logic components can be simulated and tested before integration. The individual designs are implemented and tested in parallel if possible.

The software components are developed and tested in environments equivalent to the hardware designed, and the hardware is as far as possible tested during production. The co-designed components are the last components to be implemented as they can be modified to aid in bug fixing during integration in hardware and software.

Phase Eight: Integration of Components

The hardware and the software is integrated with the programmable logic that is tested for unsuspected bugs. Should a bug appear, the bug is found and the module causing the error is retested after the bug has been corrected.

Phase Nine: Systems Integration

All modules have been tested according to the specifications made during the design phase and the requirement specifications. In this phase they are connected to each other to form the complete prototype. The complete prototype system is then tested for bugs. If all tests pass it means that the requirement specifications that where designed a solution for is met. Any tests not met, could mean that parts of the system should be redesigned.

A.1.2 Report Structure

The Camera System is documented in the report. While the project was run in phases after the method mentioned in the last section, the report is based on the three documents produced. This section treats the report that was produced during this project. The report describes the analysis, and the design. In the report the implementation of the design is also described and it presents testing results to document the prototype.

The system requirement specifications are in the documentation divided in two main sets of specifications. One set is based on demands to the circuitry and the components derived from being in a satellite. The other set is based on the needed functionalities detected in the analysis.

The analysis ends up presenting the system requirement specifications and the corresponding test specification for the requirements. The tests that were completed within the time frame of the semester is described last.

The designed hardware and the co-designed component are collected in a systems description. The chapter also describes how components are implemented to a make out platform, which the software must control. After the system description the design of the software follows. Implementation of the software is dealt with separately.

A requirement that is specified must be testable. Before the system is ready for flight, all specifications must be tested. Some of the tests can be done at the first prototype and others should be accomplished when the camera system design has been iterated. Examples of tests that can be performed at the prototype state are software functionality tests, co-hardware related tests and test estimating the power budget. However, the time constraints of the project do not allow for all tests to be accomplished.

The report is rounded by the conclusion and the perspectives. The model used invites to return to the design phase after prototyping and modify the design to make sure that the requirements that are not yet treated by the first prototype are also fulfilled before the system is finished. The perspectives will give an insight into the plans for the next iteration. A set of improvements conceived during the late design phase is also described here, so that these may be included in the next iteration.

Analysis

The analysis is divided into two subjects; an analysis of physical limits and the functionality analysis. The two subjects differ, as the first one is a research concerning the environment in space and simple mass and size requirements from the satellite. The second subject is the functionalities that can be expected from the camera system; this is areas like image handling, configurability and size of images. The work required to analyze problems regarding each of the two subjects. The first subject is analyzed using classical tools like information gathering and analysis, building on the existing research to find the more specific knowledge needed for defining specifications.

The second subject requires the developers not only to use classic tools of information gathering but also to use development tools such as “use case” analysis and temporary interface documents to control which areas that is to be investigated further. The analysis ends up with the requirement specification containing all necessary specifications for designing the camera system.

Object Oriented Designing

The software is designed using an object oriented design model. The functionalities are collected in objects that perform a task that other objects can make use of. The objects are based on different tasks located through the use cases and the requirement specifications. A number of objects are in use to form the software, but only the most significant are described in the report. Objects for communication and debug are not important to the flow of the software. Therefore, communication protocols are specified in the analysis and debug facilities are developed ad-hoc.

Attempts where done to describe the software using traditional flowcharts but in the end the complexity of the software required a better language to avoid misunderstandings. So to be able to describe the software, the Unified Modeling Language was introduced to the project as it provides tools that can be combined with the SSU lectures to form a complete description of the software design.

A combination of the UML activity state charts (activity diagrams) and a process overview from the SSU lectures hands over graphical interpretations of the designed system.

The main data structures and memory layouts combined with the state charts containing names and descriptions of arguments and returns allow the programmers to complete functions and complete modules without having to know the entire code. From this philosophy a work schedule is formed that divides the work flow into two different phases. The implementation of all objects are done bottom to top, meaning that functions and modules of the threads are coded and tested before the overall flow is implemented. The design on the other hand is created top to bottom as it is described in the report.

The procedure ensures that the simple functions needed by an object is well tested before the object is completed and put in the correct environment, where as the design starts out defining the objects, and then afterward specify the flow of the functions.

In the report the implementation of the software is described for the elements that was not already pinned out in the design. This is the lowest level in the design, but important functionality lays in this level. This is functions like the communication protocol with the images sensor, the compression algorithm, and similar functions.

A.1.3 Tools Developed for Creating the Project

To be able to implement the design a variety of tools where developed or modified to allow debug and testing. This section gives a short description of the tools used.

AAUSAT-II OBC Running eCos

To get a feeling with eCos before configuring it to the camera system an OBC prototype was borrowed from AAUSAT-II. This was done to have a platform to develop the software on while waiting for the PCB to arrive. Thereby, the project group was provided with a compilable version of eCos that was configured for a target that was comparable with the minimum system of the camera system.

Debug Thread

The running version of eCos had implemented a debugging thread. The thread is running as the only thread at highest priority making it possible for all other applications to call the debug facilities at all times. The debug thread is responsible for handling the communications on the RS-232, and allows for debug texts printing text strings and variables as ASCII on the RS-232. Using a terminal allows for reading the printouts.

The debug thread also has a build in command handling system that is easy to modify. This allows for programmers to send commands to the camera system for running functions. Using semaphores threads can in this way be stepped through by the debug facility.

Remote Programming and Reset Interface

To be able to program the OBC from a remote location, a terminal was connected to the RS-232 interface that functioned as a hardware link to the network. The debug thread interface could then be controlled from a ssh link to the terminal.

This allowed for uploading new software to the OBC and afterward run it on the OBC. However, when debugging programs, errors can occur, and to avoid the hardware to end up in an endless loop or similar problems that cannot be dealt with from the debugging thread, a remote reset circuit was connected to the supply lines. This allowed for turning the camera system off and on from any remote location using the parallel port interface of the terminal.

HSNTest

To be able to test the HSN interface on the CAN, the AAUSAT-II team has created a windows application titled HSNTest. It utilizes an USB to CAN converter to transmit and receive HSN packets. HSNTest is modified by the project group to invoke commands for the camera system as well as AAUSAT-II commands. As so the commands found in the user manual in Appendix A.10, page 163, is in all tests transmitted from the HSNTest software ensuring that the HSN protocol is followed.

A.2 The CAN Bus

The Controller Area Network (CAN) was originally developed for vehicles and is by many regarded one of the more robust serial buses. CAN is an industrial bus standard. Therefore, a wide variety of CAN transceivers are available in versions that should be suitable for space. The bus standard for CAN defines a two layer protocol consisting of a physical layer and a data link layer that when combined makes up the interface for a node on a CAN network, as can be seen on Figure A.2. There is no requirements for a master/slave relation on the bus, since the individual priority of every node can be defined in the software of the individual node. A node is the common term for a unit connected to a bus and is able to both receive and transmit data. There is no external way to control which node is to transmit on the bus, however on the CAN no data is lost even if there should occur data collisions. Transmissions are simply delayed on the node that is transmitting with the lowest priority. Both layers of the network are described in this section, to provide an understanding of the CAN.

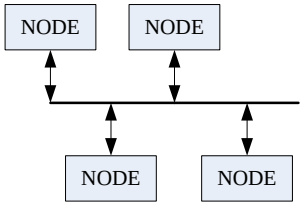


Figure A.2: *A network on the CAN bus is consisting of nodes connected to the bus, and the nodes individual priority on the bus can be encoded using the CAN protocol.*

A.2.1 Data Link Layer

When transmitting data on the CAN network, the data is formatted using a message framing. On AAUSAT-II the implemented CAN version was 2.0B. The framing for this version is illustrated on Figure A.3. All nodes on the bus will receive the transmitted bits when the CAN is active, as so all of them can decode the message, if no node verifies the transmitted data, the message is automatically resent by the controller. The origin on the data is recognized from the identifier fields. When this is determined it is followed by the data size information field, followed by the actual data field, and some acknowledgment of the message.

All messages starts with a Start Of Frame bit, that is set to inform the nodes that the bus is now active. A node must always check to see if the bus is active before trying to transmit a message. If the bus is busy, it will wait for the transmitting node to complete its message before trying to transmit a new message [Bosch, 1991, p. 11]. This check is possible to do at any time by checking more than 5 bits in a row for dominance, due to the bit stuffing procedure of the controllers. If more than 5 bits are recessive no node can be transmitting. The bit stuffing is implemented so that when a transmitter detects five consecutive bits of identical value in the bit stream that is to be transmitted, it automatically inserts a complementary bit in the actual transmitted bit stream. This extra bit is removed by the controller again. Bit stuffing is only applied on the first part of a message; from the CRC delimiter bit stuffing is not applied.

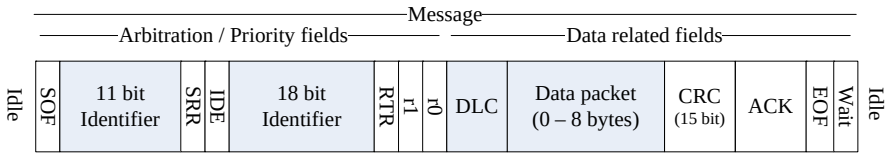


Figure A.3: *The protocol handles transmissions as a message framing, however only a few of the frames can be adjusted by software, these are marked by color [Bosch, 1991, p. 44].*

In the CAN version 2.0B the arbitration field contains two identifier bit fields. The first (the base ID) is 11 bits long for compatibility with version 2.0A. The second field (the ID Extension)

is 18 bits long, this gives a total length of 29 bits that can be adjusted by the user. The identifier fields define which node gets to send a message on the bus. All the nodes that is trying to access the bus, will try to transmit simultaneously when a transmission has just been successfully transferred. The Identifier bits are used to ensure that only one node at a time eventually ends up transmitting data. A logic '0' (dominant bit) will dominate the bus and as so the node trying to transmit a logic '1' (recessive) will be set to receive instead of transmit. The first node to transmit at logic '0' will therefore gain bus access. How this works electrically is explained in the description of the physical layer.

The priority for a message on the bus is therefore defined for each transmission by setting the identifier bits locally at each node. [Bosch, 1991, p. 44]. This ensures that when two nodes tries to transmit at the same time, the node that first transmits a bit in dominant mode, gets to complete its packet, leaving the other nodes silent, and thus in a listening mode.

Then a RTR (Remote Transmission Request) bit is send, this basically tells the receiving controller if the message sent to them contains data, or if the transmitting node is requesting more data. When RTR is set dominant, a data field will follow, if set recessive, the node transmitting is instead requesting data from the receiving node corresponding to the identifier bits sent [Bosch, 1991, p. 46].

To handle compatibility with CAN 2.0A, the 2.0B controller also transmits a SRR (Substitute Remote Request) and an IDE (IDentifier Extension) bit during the arbitration frame to inform the receivers that this message is in extended identifier format (version B). When the version A node was expecting a RTR bit to be dominant, it instead gets a SSR bit (recessive), and therefore does not expect data, instead it is set to wait and return data, the IDE bit is sent to inform the version A nodes that they should not return data [Bosch, 1991, p. 44].

The reserved bits r1 and r0, is reserved by the controller, and cannot be altered by the inputs on the controller, and will not be explained further.

The Data field also contains 2 fields that is to be handled by the user, these are the DLC (Data Length Code) and the Data frame. They are closely connected since the DLC informs the receivers using a binary value, how much data is transferred in the data frame. The data field can be from 0 to 8 bytes of data, send with MSB first, immediately after DLC [Bosch, 1991, p. 47].

Following the data field a CRC (Cyclic Redundancy Check) code is sent, it contains a checksum to provide error detection on the data transmitted. The CRC is the result of an algorithm, that is optimized to detect bit flips in the data packet. The checksum is transmitted on the bus and calculated on the receiving node. If the checksum sent and the checksum calculated matches on the receiving nodes, the following ACK bit (ACKnowledge) is set dominant by the receiver to acknowledge that the packet is received properly. This is done while the transmitter is still transferring the message this is possible because of the physical layout of the bus.

If no nodes set the ACK field dominant during transmission, the transmitting node retransmits the entire message including the arbitration frame [Bosch, 1991, p. 48].

The EOF (End Of Frame) is set recessive in 7 bits. An error frame follows directly after the EOF; if errors have occurred. Errors can be caused by CRC error, ACK error, or even a timing error, where the timing error only occurs if the clock frequency of a node has drifted off key by more than half the bit length [Bosch, 1991, p. 51].

The wait frame is an empty frame that is included to ensure that the nodes are ready for a new transmission before the bus is released [Bosch, 1991, p. 44].

To handle the communication on the CAN bus, the HSN protocol is implemented on AAUSAT-II and the camera system to handle data transmissions; this protocol is explained in Section 2.6, page 38.

A.2.2 Physical Layer

The CAN is based on two balanced wires (CANH and CANL). These carry data in form of the NRZ (Non Return to Zero) encoding standard [Bosch, 1991, p. 58]. NRZ is a simple way of encoding, where there is nothing to mark the beginning and end of a bit. Instead a local clock is used to determine when the DC level of the bus is to be interpreted as a logic state, this is illustrated on Figure A.4. NRZ encoding requires that the bus output can hold a DC level while a series of similar bits are transferred. To ensure this the CAN transceiver is equipped with a DC regulated interface

for the circuit labeled RXD and TXD, or simply RX/TX, this is the actual pin connections to the circuits. The actual CAN bus, however, does not rely on a rigid DC value.

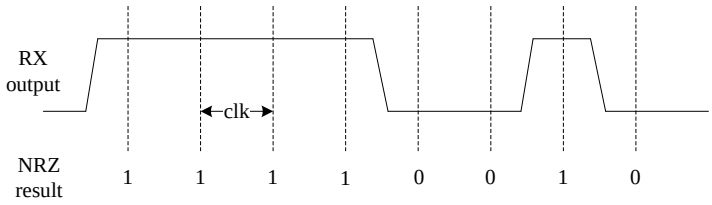


Figure A.4: *The NRZ encoding principle as a result of the output voltage on RX and the clock frequency.*

A CAN transceiver is the binding between the physical CAN bus and the CAN controller. It takes a rigid DC logic input on pin (TX) and can produce a similar logic output on pin (RX). TX/RX is in the transceiver translated to produce a dominant or recessive state for the node on the bus. If TX is set as a logic '1' both CAN lines, CANL and CANH, is set to high impedance mode (recessive mode), however, if TX is logic '0' the CANH will produce V_{CC}, and CANL will be set to GND (dominant mode). This state relation on the nodes handles priority of the bits in the identifier frame, it is this coupling that allows the arbitrary collision control [Bosch, 1991, p. 7]. A circuit equivalent for the transceivers on a small network can be seen on Figure A.5. RX will have

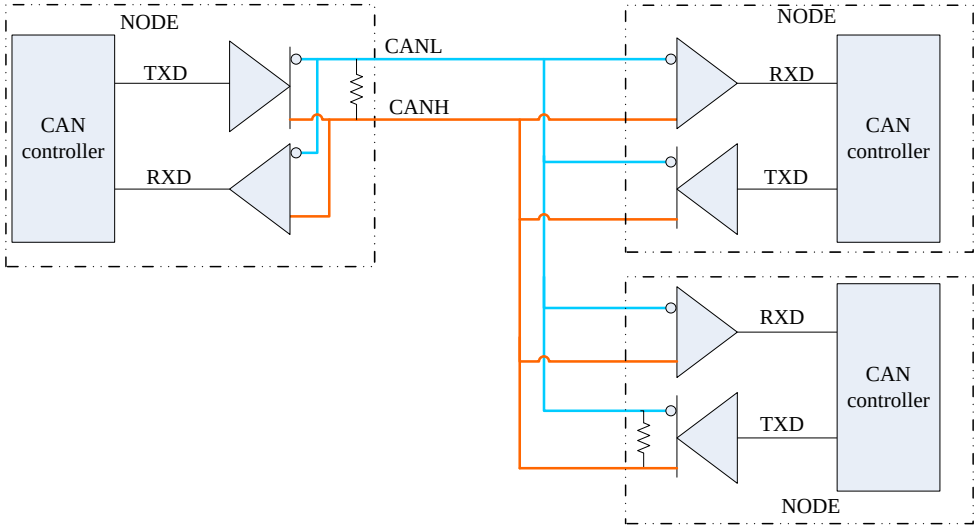


Figure A.5: *The CAN bus nodes circuit schematic. Termination resistors are only needed in the ends of the bus, their function is to match the impedance of the cable and through this reducing reflections.*

a logic '1' when the CAN bus is in recessive mode, and a logic '0' when the bus is in dominant mode. Listening on RX will therefore tell the CAN controller of the node if anything is currently transmitted on the bus. The relation between RX/TX and CANH/L can be seen on Figure A.6. Notice that the bus acts solely on the voltage difference between the two wires. This ensures that most of the electrical noise from the environment is filtered out, since it usually would appear as a common-mode signal on both CAN lines, and will therefore not alter the voltage difference. All nodes are equipped with an optional termination resistor, this resistor ensures that reflections due to line ends are minimized. A node located in the middle of the bus is not needed to have a termination resistor, since no reflections occurs here, according to the specifications for industrial use of CAN, the resistor is set to 124 Ω, this matches the characteristic impedance for a twisted pair cable [CiA, 1994, p. 3]. When the termination resistors impedance matches the cables impedance reflections are minimized.

Synchronization is an open issue, since every node locally have to produce a clock to make the

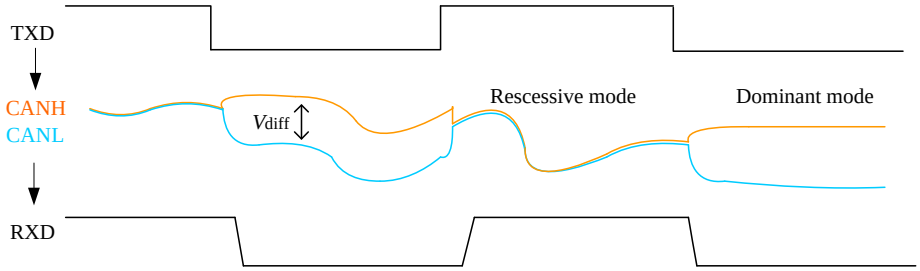


Figure A.6: *Timing for changing bits on the bus, see* Figure A.4 *for more information on the actual bit encoding [Davis, 1998].*

NRZ code possible. The tolerance of the clock frequency is given by the baud rate [Bosch, 1999, p. 6]. The NRZ is translated to the CAN standard in the transceiver, in a way that if none of the nodes sets a voltage difference on the CANH/L, the controller output (RX) will interpret the bus as transmitting high bits. Combined with the Start of Frame bit, this allows for the bus to standby in a low power mode, when no transmission occur.

A.3 Clock

A.3.1 Introduction to PLL

Phase-locked loop (PLL) is a closed-loop frequency-control system based on the phase difference between an input clock signal and a feedback clock signal of a controlled oscillator [Corporation, 2006]. A basic PLL Block diagram is shown on Figure A.7. The main blocks of the PLL are the

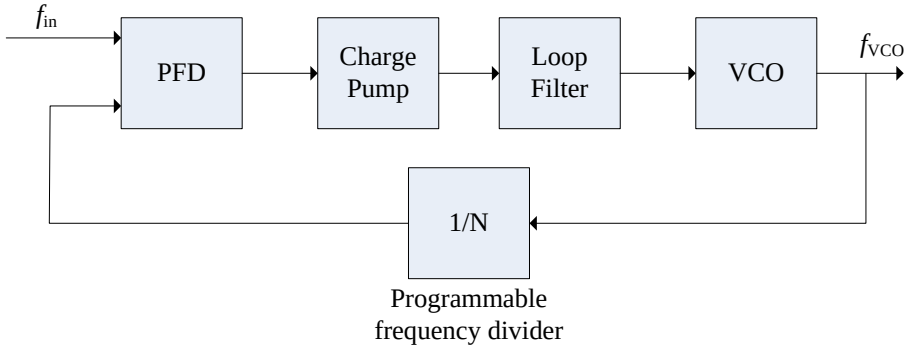


Figure A.7: *Basic PLL block schematic diagram.*

phase frequency detector (PFD), charge pump, loop filter, voltage controlled oscillator (VCO), and programmable frequency divider (feedback counter) [Corporation, 2006]. The VCO has high gain and the input from a crystal is very precise.

The PFD detects the difference in phase and frequency between a reference clock and the feedback clock inputs. Hereby, it finds out whether the VCO needs to operate at a higher or lower frequency. This control happens through the Charge Pump and Loop Filter. If the VCO should increase its frequency the charge pump drives a larger current into the loop filter, which converts the signal to a control voltage that is used to bias the VCO; then the VCO frequency increases. The output signal is sent back to the phase frequency detector via the feedback counter. The VCO stabilizes once the reference input clock and the divided feedback clock have the same phase and frequency. Thereby, the divider set the multiplier of the PLL.

A.3.2 PLL - External RC Network

The AT91SAM7A1 microcontroller has an integrated clock generator used for i.e. core clock. To multiply the input frequency to the core clock frequency a programmable PLL is used. The purpose of this section is to determine the values for the RC network used by the PLL.

From the ARM datasheet Equation (A.1), Equation (A.2), Equation (A.3), and the information in Table A.1 are collected [Atmel, 2005, p. 14].

$$0.4 < \sqrt{\frac{K_o \cdot I_p}{n \cdot (C_{103} + C_{104})}} \cdot \frac{R_{101} \cdot C_{104}}{2} < 1 \tag{A.1}$$

$$4 < \frac{C_{104}}{C_{103}} < 15 \tag{A.2}$$

$$\sqrt{\frac{K_o \cdot I_p}{n \cdot (C_{103} + C_{104})}} < \frac{\pi \cdot f_{CKR}}{5} \tag{A.3}$$

Specific values for VCO gain and CHP current are unknown, thus typical values from Table A.1 are used. The core clock frequency is set to 40 MHz using a 8 MHz crystal. In this way it follows that the division ratio n, i.e. PLL multiplication factor, is 5.

Code	Parameter	Conditions	Min	Typ	Max	Unit
f_{CKR}	Input frequency		0.02		30	MHz
n	Division ratio		1:1		1:1024	
K_o	VCO gain		65	105	172	MHz/V
I_P	CHP current		50	350	800	µA

Table A.1: *PLL Characteristics [Atmel, 2005, p. 15].*

Starting from the values together with the equations from the datasheet, Equation (A.4), Equation (A.5), and Equation (A.6) can be set up. The optimum value of Equation (A.1) is 0.707 [Atmel, 2005, p. 14].

$$\sqrt{\frac{105\ \text{MHz/V} \cdot 350\ \mu\text{A}}{5 \cdot (C_{103} + C_{104})}} \cdot \frac{R_{101} \cdot C_{104}}{2} = 0.707 \tag{A.4}$$

$$\frac{C_{104}}{C_{103}} = \frac{4 + 15}{2} \tag{A.5}$$

$$\sqrt{\frac{105\ \text{MHz/V} \cdot 350\ \mu\text{A}}{5 \cdot (C_{103} + C_{104})}} < \frac{\pi \cdot 8\ \text{MHz}}{5} \tag{A.6}$$

Solving Equation (A.4), Equation (A.5), and Equation (A.6) by letting "<" in Equation (A.6) be interpreted as the left expression is one tenth the value of the right expression the values in Table A.2 are obtained. In the table it is also shown, which values are chosen due to obtain standard component values.

Component	**Calculated value**	**Chosen value**
R_{101}	107 Ω	120 Ω
C_{103}	2.77 nF	2.2 nF
C_{104}	26.32 nF	22 nF

Table A.2: *Values for the external PLL RC network.*

A.3.3 Capacitors Connected to the Oscillator Pads

The capacitors C_{101} and C_{102} connected to the crystal, MA-506, are calculated as shown in the ARM datasheet [Atmel, 2005, p. 14]. The load capacitance is according to the datasheet between 10 pF and ∞ F [Epson, 1998]. It is chosen to let C_{101} and C_{102} have identical values, which are calculated starting form the assumption that the load capacitance is 20 pF. The calculation is shown in Equation (A.7).

$$\begin{aligned} C_{\text{load}} &= \frac{C_1 \cdot C_2}{C_1 + C_2} \\ 20\ \text{pF} &= \frac{(C_1)^2}{2C_1} \\ C_1 &= 40\ \text{pF} \end{aligned} \tag{A.7}$$

where:

C_{load} is the load capacitance from the crystal [F].

C_{101} and C_{102} are the capacitors connected by the crystal [F].

The nearest available capacitance value is chosen, which means that C_{101} and C_{102} are 47 pF.

A.4 Determination of Resistor Values

The purpose of this section is to determine resistor values for the resistors found in Table A.3. Exact locations of the resistors can be found on the CD in the Schematics folder and on page 195.

Resistor name	Pin	Location	Type
R_{106}	$\overline{\text{CS}}$	FPGA - FLASH	Pull-up
R_{107}	$\overline{\text{CE}}1$	FPGA - RAM1	Pull-up
R_{108}	$\overline{\text{CE}}1$	FPGA - RAM2	Pull-up
R_{109}	$\overline{\text{CE}}1$	FPGA - RAM3	Pull-up
R_{113}	STANDBY	FPGA - image sensor	Pull-up
R_{115}	GSHT_CTL	Image sensor	Pull-down
R_{116}	TXD1	ARM - MAX3232	Pull-up
R_{126}	$\overline{\text{RST}}$	FPGA - ARM and FLASH	Pull-down
R_{127}	$\overline{\text{RST}}$	FPGA - image sensor	Pull-down

Table A.3: *Resistors to be determined.*

A.4.1 Resistor Values for Chip Select Pins

RAMs and FLASH all have pull-up resistors at their Chip Enable or Chip Select pins. The purpose of these resistors are to ensure that the chips are not selected by accident during initializing of the FPGA; the resistors are R_{106}, R_{107}, R_{108} and R_{109}. The pins of the FPGA are floating during programming but are kept HIGH or LOW afterward.

To determine the value of a pull-up resistor, two things are to be calculated; the absolute maximum value of the resistor and the allowed minimum value. The reason for this is that even though the larger resistors value would result in a lower power usage, it would also let the result be more sensitive to noise. Therefore, a suitable value in the middle of the limits should be found. In this section the pull-up resistor, R_{107}, between the FPGA and the RAM is determined, as it appears that the RAM makes harder requirements than FLASH.

The input load currents of both the FLASH and RAM are ± 1 µA and the output leakage current of the FPGA can reach ± 10 µA [AMD, 2005, p. 40] [Cypress, 2005, p. 3] [Atmel, 2005, p. 55]. This gives a worst case of 11 µA pulling the voltage downwards. The input must be 2.2 V or higher to ensure that the RAM is not active (1.9 V for FLASH). The pull-up resistor should be lower than calculated in Equation (A.8).

$$\begin{aligned} R_{107} \cdot I_{\text{Lmax}} &< V_{\text{CC}} - V_{\text{IHmin}} \\ R_{107} &< \frac{V_{\text{CC}} - V_{\text{IHmin}}}{I_{\text{Lmax}}} \\ R_{107} &< \frac{3.3\text{ V} - 2.2\text{ V}}{11\text{ µA}} \\ R_{107} &< 100\text{ k}\Omega \end{aligned} \tag{A.8}$$

where:

I_{Lmax} is the total maximum leakage current of all the connected chips [A].

V_{IHmin} is the minimum HIGH input voltage of the RAM [V].

A resistor lower than 100 kΩ is required if the voltage level is to be correct. The lowest allowed resistor is found by looking at the LOW voltage required.

To calculate the minimal allowed resistor value for R_{107}, the output resistance of the FPGA, R_{N}, when the output is LOW, is calculated. The FPGA can in the selected configuration deliver up to 12 mA at a maximum voltage of 0.4 V. The output resistor for the FPGA is calculated in Equation (A.9).

$$\begin{aligned} R_{\mathrm{N}} &= \frac{V_{\mathrm{OLmax}}}{I_{\mathrm{OLmax}}} \\ R_{\mathrm{N}} &= \frac{0.4\ \mathrm{V}}{12\ \mathrm{mA}} \\ R_{\mathrm{N}} &= 33\frac{1}{3}\ \Omega \end{aligned} \tag{A.9}$$

where:
R_{N} is the output resistance when the state is LOW [Ω].
V_{OLmax} is the maximum output voltage when the state is LOW [V].
I_{OLmax} is the maximum output current when the state is LOW [V].

The RAM possesses the most strictly requirements to the input voltage; it will interpret a voltage below 0.4 V as a logic zero, so the voltage must not be higher than this if the chip should be enabled. The smallest allowable value of the resistor is calculated in Equation (A.10).

$$\begin{aligned} V_{\mathrm{IL}} &< \frac{R_{\mathrm{N}}}{R_{\mathrm{N}} + R_{107}} \cdot V_{\mathrm{CC}} \\ 0.4\ \mathrm{V} &< \frac{33\frac{1}{3}\ \Omega}{33\frac{1}{3}\ \Omega + R_{107}} \cdot 3.3\ \mathrm{V} \\ R_{107} &> 242\ \Omega \end{aligned} \tag{A.10}$$

A resistor that small would draw a lot of power and increase the fall time significantly. This is due to the discharging of the input capacitance that can be calculated as shown in Equation (A.11).

$$\begin{aligned} V_{\mathrm{I}}(t) &= V_{\mathrm{I}}(\infty) + \left(V_{\mathrm{I}}(0^+) - V_{\mathrm{I}}(\infty)\right) \cdot \mathrm{e}^{\frac{-t}{\tau}} \\ V_{\mathrm{I}}(t) &= \frac{R_{\mathrm{N}}}{R_{\mathrm{N}} + R_{107}} \cdot V_{\mathrm{CC}} + \left(V_{\mathrm{I}}(0^+) - \frac{R_{\mathrm{N}}}{R_{\mathrm{N}} + R_{107}} \cdot V_{\mathrm{CC}}\right) \cdot \mathrm{e}^{\frac{-t}{R_{\mathrm{N}} || R_{107} \cdot C_{\mathrm{L}}}} \quad [\mathrm{V}] \end{aligned} \tag{A.11}$$

where:
$V_{\mathrm{I}}(t)$ is the FLASH input voltage at time t [V].
C_{L} is the load capacitance of the FLASH, which is maximum 7.5 pF [F].
τ is the time constant [s].

As calculated in Equation (A.10) and Equation (A.8), R_{107} must be in the interval 242 Ω to 100 kΩ. The discharges of the load capacitor is affected by the pull-up resistor, thus elongating the fall time. It is desirable that this influence can be neglected, but at the same time, the circuit must not be too sensitive to noise. Thus R_{107} is chosen to be 10 kΩ, which gives a fall time from HIGH to LOW logic level not faster than shown in Equation (A.12).

$$\begin{aligned} 0.4\ \mathrm{V} &= \frac{33\frac{1}{3}\ \Omega}{10\ \mathrm{k}\Omega + 33\frac{1}{3}\ \Omega} \cdot 3.3\ \mathrm{V} + \left(3.3\ \mathrm{V} - \frac{33\frac{1}{3}\ \Omega}{10\ \mathrm{k}\Omega + 33\frac{1}{3}\ \Omega} \cdot 3.3\ \mathrm{V}\right) \cdot \mathrm{e}^{\frac{-t}{33\frac{1}{3}\ \Omega || 10\ \mathrm{k}\Omega \cdot 7.5\ \mathrm{pF}}} \\ t &= 0.35\ \mathrm{ns} \end{aligned} \tag{A.12}$$

where:
t is the worst-case fall time, i.e. falling from $V_{\mathrm{I}} = 3.3$ V to $V_{\mathrm{I}} = 0.8$ V [s].

The fall time is only a fraction of a clock cycle, which is 25 ns, and its influence can therefore be neglected. The 10 kΩ resistor value chosen for R_{107} will be used for R_{106}, R_{108} and R_{109} too.

A.4.2 Resistors Values for Remaining Connections

The resistors which values are to be chosen in this section all have in common that they are connected to inputs having high input impedance. The values for the resistors are not critical, but must be chosen, thus the logical level ensured by a resistor can be kept while the propagation delay is minimized.

The R_{113} is a pull-up resistor connected to the image sensor STANDBY pin to ensure that the image sensor is in standby during boot time. The value of the resistor is not critical and is set to 10 kΩ.

The R_{115} is a pull-down resistor connected to the image sensor GSHT_CTL pin to keep this fixated to GND. A resistor is set here to enable the prototype to be easy modified and the value is not critical as long as it is pulling down; thus it is set to 10 kΩ.

The R_{116} pull-up resistor on the TXD1 pin avoids a floating pin when the ARM is reset at boot time. The MAX3232 has a high input resistance and a 10 kΩ resistor is chosen.

The R_{126} and R_{127} are pull-down resistors enabling reset during boot time of ARM, FLASH and image sensor respectively. These values are set to 10 kΩ.

A.5 PCB Design Background

The purpose of this appendix is to provide the background for the PCB design by explaining the four noise concerns; ground bus noise, power bus noise, transmission line reflections, and crosstalk. This is explained in the next sections.

A.5.1 Ground Bus Noise

Ground noise is a result of signal return current and transients on the power supply. Transients on the power supply are the largest concern, because they cause the highest radiated emission and highest in-system noise voltages. To minimize the ground bus noise the impedance and thereby the inductance between the conductor and ground has to be minimized. Minimizing the impedance will result in lower voltage transient on the supply, when a current transient occurs. The inductance is minimized by changing the length and width of the signal conductor, which can be seen from Equation (A.13) [Ott, 1988, p. 281].

$$L = 196.85 \cdot 10^{-9} \ln\left(\frac{2\pi h}{w}\right) \qquad [\text{H/m}] \tag{A.13}$$

where:
L is the inductance of the conductor [H/m].
h is the height of the conductor [m].
w is the width of the conductor [m].

To minimize the susceptibility against electromagnetic fields the area of the current loop created by the signal conductors and ground has to be kept small. This is shown in Equation (A.14), where the noise voltage created by an electromagnetic field is calculated [Ebert, 2006, p. 6].

$$v_{\text{N}} = \text{j}\omega AB\cos(\theta) \qquad [\text{V}] \tag{A.14}$$

where:
v_{N} is the noise voltage [V].
ω is the angular frequency [rad/s].
A is the area [m^2].
B is the electromagnetic field [Wb/m^2].
θ is the angle between the electromagnetic field and the loop [rad].

Equation (A.13) and Equation (A.14) show that the signal conductors have to be routed close to ground and have to be as short as possible to minimize ground bus noise.

A.5.2 Power Bus Noise

When high speed digital circuitry switches logic value a large supply current is drawn, which makes a large noise voltage on V_{CC}. This noise can be minimized by using a power plane and decoupling capacitors, but the power bus design is not as important as the ground design [Ott, 1988, p. 286].

The decoupling capacitors deliver a high frequency current when a digital circuit switches and makes a transient current. Therefore, the decoupling capacitors should be mounted as close to the component as possible and have a small inductance, because this gives a high resonance frequency, as shown in Equation (A.15). Capacitors suitable for this task are disk ceramic capacitors.

$$f = \frac{1}{2\pi\sqrt{LC}} \qquad [\text{Hz}] \tag{A.15}$$

where:
f is the resonance frequency [Hz].
C is the capacitance of the decoupling capacitor [F].
L is the inductance of the decoupling capacitor [H].

The decoupling capacitors should be able to deliver the entire current, when the digital circuit switches and to do that the size of a decoupling capacitor can be calculated as shown in Equation (A.16) [Ott, 1988, p. 289].

$$C = \frac{dI\,dt}{dV} \quad [\text{F}] \tag{A.16}$$

where:
C is the capacitance of the decoupling capacitor [F].
dI is the transient current [A].
dt is the duration of the transient [s].
dV is the transient voltage drop in the supply [V].

When choosing decoupling capacitors the minimum size of capacitor should be chosen, because larger capacitors have a larger inductance. Surface mounted capacitors are preferable, because surface mounted capacitors have a small inductance [Ott, 1988, p. 293].

To avoid differential mode radiation a filter consisting of a capacitor and a ferrite bead should be placed where the power enters the PCB [Ott, 1988, p. 308]. This filter decouples noise voltages in series with the supply i.e. differential mode. To avoid common-mode radiation common-mode chokes, which increases the common-mode impedance of the cable, should be connected to the power lines [Ott, 1988, p. 317]. A common-mode choke is a component as shown in Figure A.8, which increases impedance, when common-mode noise occurs.

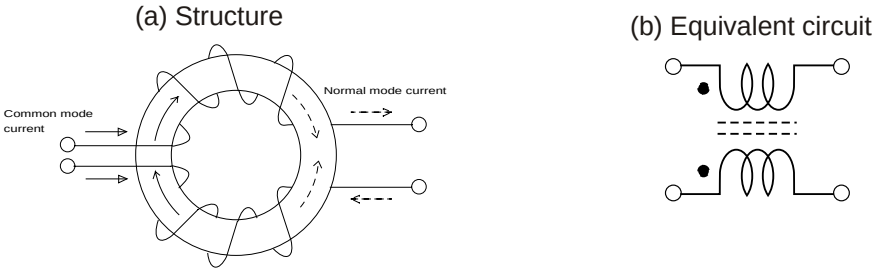


Figure A.8: *Shows a common-mode choke and its equivalent circuit [muRata, 1998, p. 2].*

A.5.3 Transmission Line Reflections

Transmission line reflections causes signals to be reflected at the load and generator due to impedans mismatch. To prevent problems due to reflections in the signal conductors, these have to be shorter than the critical length or be terminated. The critical length is exceeded when the travel time for a wave from one end and back is greater than the rise time or fall time for the pulse. This can be calculated as shown in Equation (A.17) [Bak-Jensen, 2006, p. 4].

$$\begin{aligned} \tau_{\text{trans}} &= \frac{1}{v} \\ l &= \frac{\tau_{\text{r}}}{2\tau_{\text{trans}}} \quad [\text{m}] \end{aligned} \tag{A.17}$$

where:
τ_{trans} is the transmission time [s/m].
v is the signal speed [m/s].
τ_{r} is the rise time [s].
l is the critical length [m].

When a wire is longer than the critical length, the wire has to be terminated. This is done to minimize the reflection. The reflection coefficients can be calculated as shown is Equation (A.18) and Equation (A.19).

$$K_{\mathrm{L}} = \frac{Z_{\mathrm{L}} - Z_0}{Z_{\mathrm{L}} + Z_0} \quad [\cdot] \tag{A.18}$$

$$K_{\mathrm{G}} = \frac{Z_{\mathrm{G}} - Z_0}{Z_{\mathrm{G}} + Z_0} \quad [\cdot] \tag{A.19}$$

where:
K_{L} is the reflection coefficient at the load $[\cdot]$.
Z_{L} is the load impedance $[\Omega]$.
Z_0 is the characteristic impedance $[\Omega]$.
K_{G} is the reflection coefficient at the generator $[\cdot]$.
Z_{G} is the generator impedance $[\Omega]$.

Equation (A.18) and Equation (A.19) equations show that $Z_{\mathrm{L}} = Z_0$ and $Z_0 = Z_{\mathrm{G}}$ should be met to avoid reflections if the length of the conductor is greater than the critical length. If this is not met impedance adjustment is necessary at the generator and load.

A.5.4 Crosstalk

Crosstalk is when a signal in one conductor is radiated into other parts of the circuit, because of capacitive and inductive coupling. Crosstalk can be minimized by minimizing the capacitive coupling and inductive coupling between the signal conductors. The desired situation is to have a capacitive coupling of 0 F, because a capacitive coupling makes an electric field. Furthermore, the inductive coupling should be of 0 H, because a inductive coupling makes a magnetic field [Ott, 1988, p. 19]. The coupling of two conductors can be illustrated as shown in Figure A.9. From

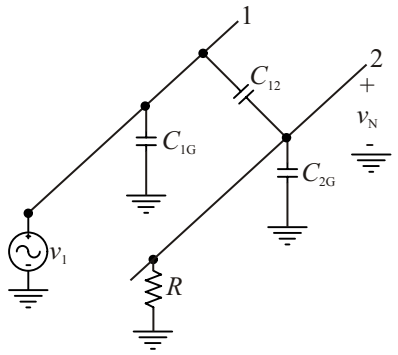


Figure A.9: *Shown the capacitive coupling between two conductors [Ott, 1988, p. 30].*

the figure the noise voltage created from the capacitive coupling on the second conductor can be calculated as shown in Equation (A.20).

$$v_{\mathrm{N}} = \mathrm{j}\omega R C_{12} v_1 \quad [\mathrm{V}] \tag{A.20}$$

where:
v_{N} is the noise voltage on conductor 2 [V].
v_1 is the generator voltage on conductor 1 [V].
C_{12} is the capacitive coupling between conductor 1 and 2 [F].
R is the resistance from the circuitry connected to conductor 2 $[\Omega]$.

The capacitive coupling between two parallel conductors can be minimized by shielding the conductors, and by proper orientation and separation of the conductors. The capacitance, dependence of the separation, can be calculated as shown in Equation (A.21) [Ott, 1988, p. 31].

$$C_{12} = \frac{\pi\varepsilon}{\ln\left(\frac{2D}{d}\right)} \qquad [\mathrm{F/m}] \tag{A.21}$$

where:
D is the distance between the two conductors [m].
d is the diameter of the conductors [m].
ε is the permeability [F/m].

Equation (A.21) shows that the effectiveness of moving the conductors apart is greatest, when the conductors are close to each other, because the ratio between the diameter and distance is logarithmic.

The size of the inductive coupling can be calculated as shown in Equation (A.22) [Ebert, 2006, p. 20].

$$L_{21} = \frac{\mu}{\pi} \ln\left(\frac{2D}{d}\right) \qquad \left[\frac{\mathrm{H}}{\mathrm{m}}\right] \tag{A.22}$$

where:
μ is the permittivity [H/m].

Equation (A.20), Equation (A.21), and Equation (A.21) show that separation of conductors is desirable, if crosstalk should be minimized.

A.6 eCos - The Chosen Operative System

eCos is chosen as the base for the functionality of the camera system. The operating system is described in this section to provide a basic understanding of the functionality it provides.

eCos is implemented on the AAUSAT-II OBC to create the platform for CDH implementation. eCos delivers basic functionality such as mailboxes, shared memory access, dynamic memory allocation, and a scheduler to running threads. It is by the AAUSAT-II OBC developers stripped for all extra unnecessary functionality [03gr731, 2003]. The reduction of eCos resulted in a very small amount of code.

The features that were considered necessary by the CDH group were later implemented or modified to control the specific hardware; this modified version of eCos is the basic software platform that is used in this project [04gr720, 2004]. An overview of the platform is described in this section.

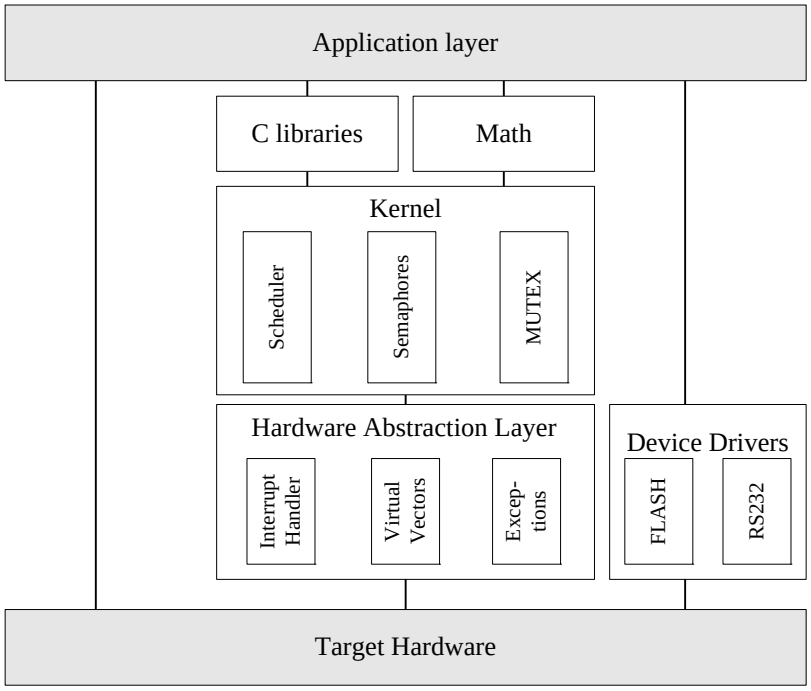


Figure A.10: *The layers of eCos as they relate to the hardware. The figure is a generel interpretation of eCos and is inspired from [Massa, 2002, p. 9].*

On Figure A.10 an overview of an eCos implementation is shown. eCos includes general C libraries and provides access to Math libraries. These are provided as header files and allows to use all the basic C functions such as malloc() and free(). The memory pool where these function exist, can be configured in the configuration files.

Scheduler in eCos

eCos has a scheduler to control the dispatching of threads. The purpose of a scheduler is to allow different threads to gain processor time. Different strategies exists for scheduler implementations; eCos features three different operation modes.

A scheduler can be thought of as two linked lists, as illustrated on Figure A.11. The two linked lists correspond to two possible states of the threads. Threads can be either idle or active. Idle threads are threads that are waiting for something to happen. The scheduler is activated after a certain interval to update the two linked lists if an event occurs that the threads are waiting for. When the lists are updated the scheduler takes the appropriate action in distributing processor time to a thread.

The eCos scheduler uses the priority of threads to control the dispatching. The scheduler can at any time be locked from within a thread ensuring that the thread is allowed to keep on running until the scheduler is unlocked again by the thread. This can be useful if a thread needs the extra

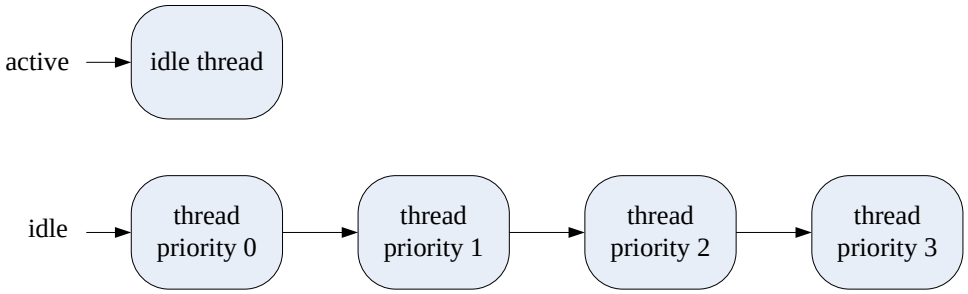


Figure A.11: *Two linked lists; an idle linked list of threads and a run queue.*

processing resources for a short while. The highest runtime priority is assigned to the lowest priority number.

The eCos scheduler can be run in three modes:

- Bitmap mode
- Multi level scheduler queue mode
- Lottery mode

The bitmap mode, illustrated on Figure A.12 uses thread priority to sort the run queue. The run queue here dispatches to the highest priority in the queue at all times. Bitmap mode requires that each thread is assigned a unique priority. Threads with lower priority never gets processor time before all higher priority threads have entered an idle list (a waiting state). The multi level

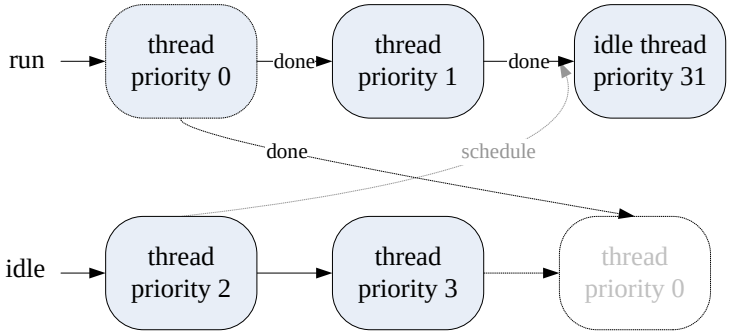


Figure A.12: *A scheduler in bitmap mode consists of two linked lists; an idle linked list of threads and a run queue. A thread in a queue must return to idle mode before the next thread is run. The run queue is always sorted by priority, and no threads may have the same priority.*

scheduler queue mode is much like an extension upon the bitmap. The run queue is still sorted after priority, but threads are allowed to have the same priority, if this happens they are forced to share processor time between them within the priority. Still all threads with a higher priority must be moved to the idle queue before a lower priority thread is given time. Thereby, the threads with higher priority still are assigned more processing resources although they share priority number. This concept is illustrated on Figure A.13.

The lottery mode is as the name implies much like a lottery. Each thread in the run queue owns tickets and a random function chooses a ticket from a pool. The thread owning the pooled ticket is the one is to gain processor time. The priority is controlled by how many tickets the thread owns in the lottery.

Interrupt Handling in eCos

Usually when interrupts are generated by the hardware it calls an exception vector pointing at an ISR (Interrupt Service Routine). The actual implementation of the ISRs is very hardware dependent and will vary among the different processor types. eCos is configured to the specific hardware and

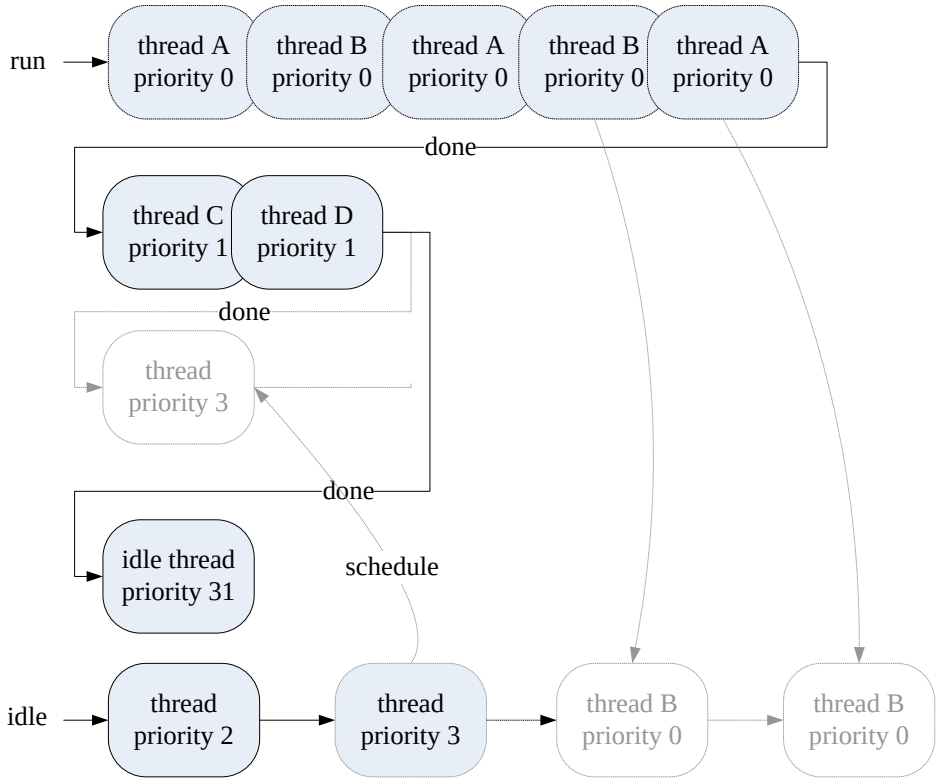


Figure A.13: *A multi level queue scheduler: Two linked lists; an idle linked list of threads and a run queue. All the threads with an equal priority are gathered in the same list. All thread in a high level priority list must return to idle mode before the next level of priority list of threads are run. Within a list the processing power to the threads are time sliced, thus the threads sharing processor time equally.*

therefore the specific ISRs. eCos uses this low level interrupt service routine to create an extra layer of interrupt handling, providing a uniform interrupt handling for different processor types.

When an ISR is activated from the interrupt exception vectors, the processor runs the code in the routine immediately, unless other ISRs with a higher priority are currently being processed. The time a low priority ISRs have to wait before being processed because of this interrupt scheduling is defined as interrupt latency.

To reduce the interrupt latency in eCos based systems, the actual code in an ISR is kept as reduced as possible. Only high priority interrupts that needs extremely fast processing are to be ever processed as an ISR, and only if the system is properly set up. This is especially important if eCos runs on a processor with a single vector for all interrupts. In the ARM processor though multiple vectors are already used by the processor itself. eCos allows for a very limited code in the ISRs, by letting the ISRs enable a DSR (Deferred Service Routine) and then afterward end.

The DSR is a service routine that in contrast to the ISR waits to process the interrupt till the scheduler is ready for it. As so, interrupts are quickly removed from the exception vectors designated by the hardware, and spread out to the DSRs in the run queue threads. Interrupts handled through the DSR system can in this way be processed by quite complex software without disturbing the processing of smaller and faster interrupt routines. However, the code in an ISR will naturally always be run at a higher priority than a DSR [Massa, 2002].

One advantage of the DSR, is that it can use the functionality of the eCos kernel, where as ISR cannot. The scheduler itself uses a clocked ISR to call itself, ensuring that it, when enabled, is run frequently. The DSR has a high priority in the scheduler and is therefore often run immediately after the ISR has ended, unless the scheduler has other DSRs of higher priority waiting, or a thread has locked the scheduler. Tempting as it might seem, DSRs cannot contain blocking calls like semaphores.

However, if a thread has locked the scheduler even the DSR is set to wait execution till the

scheduler is released again. Locking the scheduler from a thread in this matter means that only ISRs are run, these are, however, not intended to be used by the application programmers unless there are extreme real time demands to a specific routine.

A.6.1 Semaphores

Basically, a semaphore is a general way of letting threads know if any predefined event happens. A semaphore is more specific used for counting exactly how many times something happens without having been processed. When a thread then later processes the event, the semaphore is counted downwards again. A flow of this is illustrated on Figure A.14.

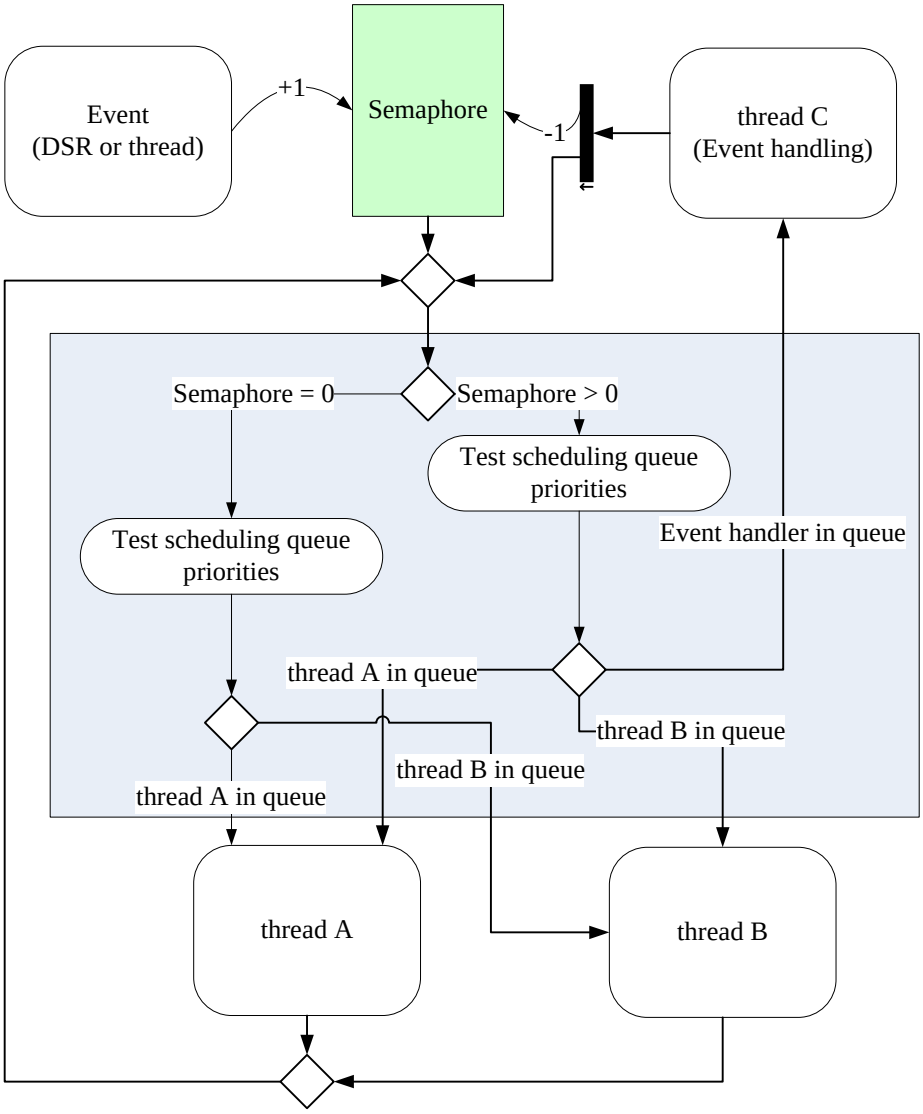


Figure A.14: *A very simplified scheduler here found in the blue box is checking the semaphores to find out if the threads waiting for it is allowed to process the event. An event creater adds one to the semaphore and when the event handler is activated it subtracts one from the semaphore.*

In eCos this is implemented so that both hardware events and other threads can produce a new event in a semaphore by incrementing the semaphore counter. Threads that processes the event decrement the semaphore counter again. [Massa, 2002, p. 40]

If a semaphore counter ends up at being zero, it is interpreted by eCos as if all the events counted have been processed by threads. When this occurs threads that are checking for unprocessed events in the semaphore counter are kept in a wait state by the kernel and will not be allowed to do anything before the counter is incremented by something.

Should more than one thread be assigned to process the events listed in the semaphore, they are put in a FIFO like queue so that the first thread arrived in the queue is the first one unlocked, unlocking only one thread for each event that occurs. This gives an approach to threads that can allow for the system to use very few resources on threads that are waiting for something to occur in the semaphore.

Semaphores can be used for event and interrupt handling or even to handle access to the protected memory areas. However, eCos provides alternate ways of controlling the memory areas that are recommended by the developers to use instead; for this task a MUTEX function is implemented.

A.6.2 MUTEX

MUTual EXception handling in eCos allows the programmer to access shared memory within a thread without having to consider whether other threads need access to the same memory. The problem of shared memory is that it can be changed by multiple sources at all times. If two or more threads needs to update a variable, the standard procedure within the thread would be to read the content of the variable, do something to it, and then afterward store it at the variables location.

Having one thread adding a number to a variable, e.g. +1, and afterward having another thread remove the same digit, e.g. -1, would logically produce a zero ($0 + 1 - 1 = 0$). This passes fine as long as the scheduler lets the threads share processor time successively.

If the "addition" thread, however, is shifted out by the scheduler before the addition is saved, but after having read the variable it would have read a zero. Suppose it is the "reduction" thread that the scheduler now activates. It would read the old value of the variable (zero), reduce it to -1, and then finish. After some time the "addition" thread is set to continue where it stopped, remember that it read a zero and added a +1, now it would save a +1 in the variable and the calculation fails ($0 - 1 + 1 \neq +1$).

Such flaw would be able to go undetected since the error is not necessarily occurring very often, however, it could be critical to some applications.

A MUTEX locks the memory before reading the variable, and unlocks it when saved. Other threads that tries to gain access to the variable are set to wait for the variable to be unlocked. The danger of using a MUTEX is that in eCos only the thread that locked a variable can unlock it again. As so the programmer must remember to unlock the variable when done. The previously described scenario would now run as follows.

The "addition" thread gains access to the variable, reads a zero and is shifted out by the scheduler. The "reducing" thread tries to gain access to the variable, but is set to wait. The "addition" thread now completes the addition and unlocks the variable. When the "reducing" thread now is allowed access again, the addition was stored and now the variable would perform as it should ($+1 - 1 = 0$).

A.6.3 Device Drivers

Device drivers provides the application programmer with an easy to use access to hardware. Some of these come shipped with eCos, but most of them should be specifically coded later on. The functionality of the device drivers are often to access the interrupt DSR (Delayed Service Routines) and wait for events. If it is a serial communication the driver will often buffer the packet, handle the low level protocol, and alert processing threads that data has arrived, or allow the threads to transmit data packets via the hardware. However, this will variate since it is a very hardware specific task. Device drivers should be used whenever an I/O access to a peripheral device is needed by more than a single thread.

A.7 Wait States

To setup the EBI the number of wait states necessary to read to and write from FLASH and RAM has to be calculated. The times used in the calculations are shown in Table A.4. The number of

Code	Parameter	Min	Max	Unit
ARM - read				
tr_{ADCRDV1}	Address change to read data valid		$(1 + \text{NWS}) \cdot t_{\text{cycle}} - 11.0$ ns	s
tr_{CSLRDV}	Chip select low to read data valid		$(1 + \text{NWS}) \cdot t_{\text{cycle}} - 10.5$ ns	s
tr_{OELRDV1}	Output enable low to read data valid		$(0.5 + \text{NWS}) \cdot t_{\text{cycle}} - 11.5$ ns	s
tr_{BSLRDV1}	Byte select low to read data valid		$(1 + \text{NWS}) \cdot t_{\text{cycle}} - 11.5$ ns	s
ARM - write				
tw_{WPL1}	Write pulse low		$(1 + \text{NWS}) \cdot t_{\text{cycle}} - 1.6$ ns	s
tw_{DSWH1}	Data setup time to write high		$(1 + \text{NWS}) \cdot t_{\text{cycle}} - 3.6$ ns	s
FLASH - read				
t_{ACC}	Address to Output Delay	120		ns
t_{CE}	Chip Enable to Output Delay	120		ns
t_{OE}	Output Enable to Output Delay	30		ns
FLASH - write				
t_{PWE}	Data Setup Time	45		ns
t_{WPH}	Write Pulse Width	35		ns
RAM - read				
t_{AA}	Address to Data Valid	55		ns
t_{ACE}	$\overline{\text{CE}}_1$ LOW and $\overline{\text{CE}}_2$ HIGH to Data Valid	55		ns
$t_{\overline{\text{DOE}}}$	$\overline{\text{OE}}$ LOW to Data Valid	25		ns
t_{DBE}	BLE/BHE LOW to Data Valid	55		ns
RAM - write				
t_{PWE}	$\overline{\text{WE}}$ Pulse Width	40		ns
t_{SD}	Data Set-Up to Write End	25		ns

Table A.4: *Times are fetched from the datasheets for ARM, FLASH, and RAM [Atmel, 2005, p. 44], [AMD, 2005, p. 42] [Cypress, 2005, p. 5]. The times are valid, when ARM is in standard read mode and FPGA has an input capacitance of 8 pF [XILINX, 2004, p. 55].*

wait states needed to read from FLASH with speed option 120R is calculated in Equation (A.23), Equation (A.24), and Equation (A.25).

$$\begin{aligned} 120 \text{ ns} &= (1 + \text{NWS}) \cdot t_{\text{cycle}} - 11.0 \text{ ns} \\ \text{NWS} &= \frac{120 + 11.0}{25} - 1 \\ \text{NWS} &= 4.24 \end{aligned} \tag{A.23}$$

$$\begin{aligned} 120 \text{ ns} &= (1 + \text{NWS}) \cdot t_{\text{cycle}} - 10.5 \text{ ns} \\ \text{NWS} &= \frac{120 + 10.5}{25} - 1 \\ \text{NWS} &= 4.22 \end{aligned} \tag{A.24}$$

$$
\begin{aligned}
30\ \text{ns} &= (0.5 + \text{NWS}) \cdot t_{\text{cycle}} - 11.5\ \text{ns} \\
\text{NWS} &= \frac{30 + 11.5}{25} - 0.5 \\
\text{NWS} &= 1.16
\end{aligned} \tag{A.25}
$$

The number of wait states needed to write to FLASH with speed option 120R is calculated in Equation (A.26) and Equation (A.27).

$$
\begin{aligned}
35\ \text{ns} &= (1 + \text{NWS}) \cdot t_{\text{cycle}} - 1.6\ \text{ns} \\
\text{NWS} &= \frac{35 + 1.6}{25} - 1 \\
\text{NWS} &= 0.464
\end{aligned} \tag{A.26}
$$

$$
\begin{aligned}
45\ \text{ns} &= (1 + \text{NWS}) \cdot t_{\text{cycle}} - 3.6\ \text{ns} \\
\text{NWS} &= \frac{45 + 3.6}{25} - 1 \\
\text{NWS} &= 0.944
\end{aligned} \tag{A.27}
$$

The number of wait states needed to read from RAM of 55 ns type is calculated in Equation (A.28), Equation (A.29), Equation (A.25), and Equation (A.31).

$$
\begin{aligned}
55\ \text{ns} &= (1 + \text{NWS}) \cdot t_{\text{cycle}} - 11.0\ \text{ns} \\
\text{NWS} &= \frac{55 + 11.0}{25} - 1 \\
\text{NWS} &= 1.64
\end{aligned} \tag{A.28}
$$

$$
\begin{aligned}
55\ \text{ns} &= (1 + \text{NWS}) \cdot t_{\text{cycle}} - 10.5\ \text{ns} \\
\text{NWS} &= \frac{55 + 10.5}{25} - 1 \\
\text{NWS} &= 1.62
\end{aligned} \tag{A.29}
$$

$$
\begin{aligned}
25\ \text{ns} &= (0.5 + \text{NWS}) \cdot t_{\text{cycle}} - 11.5\ \text{ns} \\
\text{NWS} &= \frac{25 + 11.5}{25} - 0.5 \\
\text{NWS} &= 0.964
\end{aligned} \tag{A.30}
$$

$$
\begin{aligned}
55\ \text{ns} &= (1 + \text{NWS}) \cdot t_{\text{cycle}} - 11.5\ \text{ns} \\
\text{NWS} &= \frac{55 + 11.5}{25} - 1 \\
\text{NWS} &= 1.66
\end{aligned} \tag{A.31}
$$

The number of wait states needed to write to RAM of 55 ns type is calculated in Equation (A.32), and Equation (A.33).

$$
\begin{aligned}
40\ \text{ns} &= (1 + \text{NWS}) \cdot t_{\text{cycle}} - 1.6\ \text{ns} \\
\text{NWS} &= \frac{25 + 1.6}{25} - 1 \\
\text{NWS} &= 0.064
\end{aligned} \tag{A.32}
$$

$$
\begin{aligned}
25\ \text{ns} &= (1 + \text{NWS}) \cdot t_{\text{cycle}} - 3.6\ \text{ns} \\
\text{NWS} &= \frac{40 + 3.6}{25} - 1 \\
\text{NWS} &= 0.744
\end{aligned} \tag{A.33}
$$

According to these calculations 5 wait states should be inserted, when communicating with FLASH and 2 wait states should be inserted, when communicating with RAM.

A.8 Waveforms on Timing Diagrams

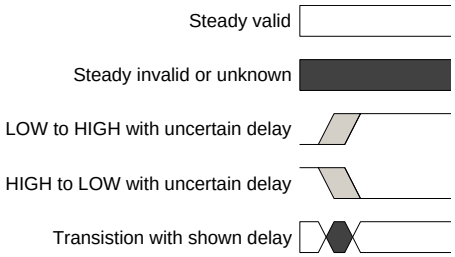

Steady valid
Steady invalid or unknown
LOW to HIGH with uncertain delay
HIGH to LOW with uncertain delay
Transistion with shown delay

Figure A.15: *Explanation of waveforms used in timing diagrams.*

The timing diagrams are used to explain sequences in the report. On the timing diagrams the state of signals are shown as waveforms, and on Figure A.15 a text explains what state each of the waveforms corresponds to.

A.9 Default Settings on the Image Sensor

This section describes the difference between default settings of the image sensor, and default settings used for this project. The settings changed should make sure that the image sensor is in snap shot mode and has the desired resolution.

To get the desired functionality the two settings described below must be set.

- **Set the image sensor in snap shot mode.**
  This is done by writing '1' in bit 8 in register 0x1E and makes it necessary to trigger the image sensor to capture an image.
- **Change the resolution of the image sensor to** 1024×1536**.**
  This is done by writing '1' in bit 0 in register 0x23 and makes the image sensor skip half the columns.

Default settings are shown in Table A.5 and are loaded into FLASH in the block reserved for these settings.

Register (Hex)	**Setting** (Binary)
0x01	0000 0000 0001 0100
0x02	0000 0000 0010 0000
0x03	0000 0101 1111 1111
0x04	0000 0111 1111 1111
0x05	0000 0000 1000 1110
0x06	0000 0000 0001 1001
0x07	0000 0000 0000 0010
0x08	0000 0000 0000 0000
0x09	0000 0110 0001 1001
0x0A	0000 0000 0000 0000
0x0B	0000 0000 0000 0000
0x0C	0000 0000 0000 0000
0x0D	0000 0000 0000 0000
0x1E	0000 0001 0000 0000
0x20	0000 0000 0000 0000
0x21	0000 0000 0000 0000
0x22	0000 0000 0000 0000
0x23	0000 0000 0000 0001
0x2B	0000 0000 0000 1000
0x2C	0000 0000 0000 1000
0x2D	0000 0000 0000 1000
0x2E	0000 0000 0000 1000
0x32	0000 0000 0000 0000
0x35	0000 0000 0000 1000
0x49	0000 0000 1010 1000
0x4B	0000 0000 0010 1000
0x5D	0010 1101 0001 0011
0x5F	0010 0011 0001 1101
0x60	0000 0000 0010 0000
0x61	0000 0000 0010 0000
0x62	0000 0000 0000 0000
0x63	0000 0000 0010 0000
0x64	0000 0000 0010 0000
0xF8	0000 0000 0000 0001

Table A.5: *Default settings loaded to the image sensor.*

A.10 User Manual for AAUSAT-III CAM

This user manual describes how to interface with the camera system.
Developed by 06gr415 in the spring semester 2006.
PCB version 1.0.

Locations of the physical connections to the PCB are shown on Figure A.16. An interface description is found in Table A.6.

A.10.1 Connections to the Camera System

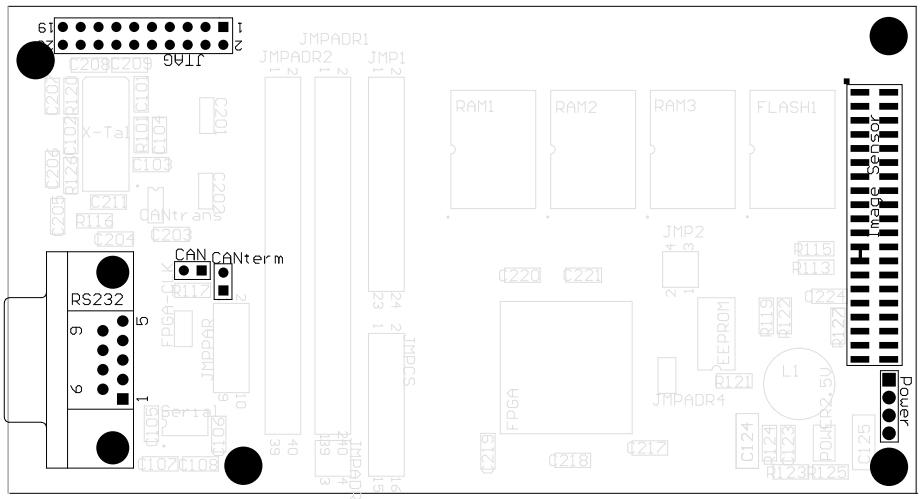


Figure A.16: *The connectors on the camera system are used for interfacing.*

Connector	Pin	Rating
POWER	1	3.3 V
	2	GND
	3	5 V
	4	2.5 V output
RS-232	5	GND
	3	TX
	2	RX
JTAG	3	TEST
	5	TDI
	7	TMS
	9	TCK
	13	TDO
	15	RST
	1,2	3.3 V output
	4,6,8,10,11,12,14,16,18,20	GND
CAN	1	CANH
	2	CANL
CANterm	1	JMP*
	2	JMP*
Image Sensor		40 pin cable for Devitech BlackEye board

Table A.6: *The physical interfaces of the circuit. *Jumper for enable CAN termination resistor.*

RS-232	38400 baud 8 bit 1 stop bit NO parity NO flow control
CAN 2.0B	125 kb/s Extended HSN protocol
Storage Temperature	-40 °C to +85 °C
Operational Temperature	0 °C to 60 °C

Table A.7: *The specifications for handling the camera system.*

Communications with the camera system are done on the CAN and follows the HSN protocol. The actual commands that can be transferred are listed in Table A.9 with the possible responses in Table A.10. The arguments that the commands and responses can be extended with are presented in Table A.11. An overview of the commands and responses are found in Table A.8.

The RS-232 is intended for debugging purposes only. The JTAG has direct connection to the ARM and indirectly to the FPGA. Through a boundary scan it should be possible to debug the FPGA as they are connected following the IEEE 1149.1 standard, though the interface has not been tested.

Command	**Arguments**	**Status**	**Return**
HSN_CMD_CAM_CDH_SETUP	*length* DATA*	**DONE** INVALID_SETTINGS FLASH_SETTINGS	
HSN_CMD_CAM_CDH_DEFAULT		**DONE** DEFAULT_ERROR	
HSN_CMD_CAM_CDH_CAPT	*time* *thumb* *raw*	**DONE** OUT_OF_SPACE SENSOR_FAILED OUT_OF_TEMPERATURE_RANGE TEMP_SENSOR_FAILED RAW_IMAGE_NOTHUMB RAW_NOIMAGE_THUMB RAW_NOIMAGE_NOTHUMB NORAW_IMAGE_THUMB NORAW_IMAGE_NOTHUMB NORAW_NOIMAGE_THUMB NOTHING_SAVED	
HSN_CMD_CAM_CDH_LIST		**DONE** NO_IMAGES_EXIST	LIST*
HSN_CMD_CAM_CDH_DEL	*img_id*	**DONE** NO_SUCH_IMAGE	
HSN_CMD_CAM_CDH_SEND	*img_id*	**DONE** NO_SUCH_IMAGE NO_SUCH_CHUNK	DATA_START* DATA** DATA_STOP*
HSN_CMD_CAM_CDH_GET_HK		**DONE** TEMP_SENSOR_FAILED	HK_DATA*
HSN_CMD_CAM_CDH_PING			PONG

Table A.8: *The list of commands with possible responses. All status messages are returned as a HSN_CMD_CDH_CAM_STATUS. All commands are defined as HSN_CMD_(Receiver)_(Sender)_(Command name). Status messages that are not DONE are defined as ERROR_(status). * Indicates that the packet contains data. ** Indicates that the packet type contains data and can be repeated.*

Command	Enum	Address	Description
HSN_CMD_CDH_CAM_PONG	1	CAM to CDH	A reply to CDH when HSN_CMD_CAM_CDH_PING is received.
HSN_CMD_CAM_CDH_PING	2	CDH to CAM	Camera replies with a PONG to CDH.
HSN_CMD_CAM_CDH_CAPT	3	CDH to CAM	Camera captures an image and stores it in FLASH of the camera.
HSN_CMD_CAM_CDH_LIST	4	CDH to CAM	Camera generates a list with info of stored images.
HSN_CMD_CAM_CDH_DEL	5	CDH to CAM	Camera deletes an image from FLASH of the camera.
HSN_CMD_CAM_CDH_SEND	6	CDH to CAM	Camera transmits image data in chunks of 10 kb.
HSN_CMD_CAM_CDH_SETUP	7	CDH to CAM	Camera will use these settings for the next image captured. When transmitting the string of registers and settings, be aware that a register may only appear once in the string and that any string longer than 34 registers of 8 bit registers with 16 bits of settings for each, is invalid. Whenever register 2 or 23 is send, both of them must appear in the stream.
HSN_CMD_CAM_CDH_DEFAULT	8	CDH to CAM	Camera will return to default settings. These are used for next image captured.
HSN_CMD_CAM_CDH_GET_HK	9	CDH to CAM	Camera will return housekeeping data. These should be put in log and transmitted with next beacon.
HSN_CMD_CDH_CAM_DATA_START	10	CAM to CDH	Image data transmission begins, img_id, *time*, and chunknumber is send to identify the data.
HSN_CMD_CDH_CAM_DATA	11	CAM to CDH	More image data is send, and more is to follow.
HSN_CMD_CDH_CAM_DATA_STOP	12	CAM to CDH	Last package that contains image data of this chunk.
HSN_CMD_CDH_CAM_STATUS	13	CAM to CDH	A standard reply to inform CDH if the camera executed the command properly.
HSN_CMD_CDH_CAM_LIST	14	CAM to CDH	A list of images is send.
HSN_CMD_CDH_CAM_HK_DATA	15	CAM to CDH	This is the return of a housekeeping request; it contains the housekeeping data.

Table A.9: *Description of the transferred commands. CAM is the label for the camera system*

Status codes	Enum	Argument	Description
DONE	0	Command	Return for successful command execution.
INVALID_SETTINGS	1	Command	Settings transferred did not comply with datasheet of image sensor.
FLASH_SETTINGS	2	Command	Current settings could not be overwritten.
OUT_OF_SPACE	3	Command	No room for new images.
SENSOR_FAILED	4	Command	The image sensor could not be initialized.
OUT_OF_TEMPERATURE_RANGE	5	Command	The temperature sensor was currently not within the allowed temperature range.
TEMP_SENSOR_FAILED	6	Command	Could not read temperature from the temperature sensor at image sensor board.
RAW_IMAGE_NOTHUMB	7	Command	A thumbnail requested could not be saved.
RAW_NOIMAGE_THUMB	8	Command	An image requested could not be saved.
RAW_NOIMAGE_NOTHUMB	9	Command	Neither the image or the thumb requested could be saved.
NORAW_IMAGE_THUMB	10	Command	The raw image could not be saved.
NORAW_IMAGE_NOTHUMB	11	Command	Neither raw or thumbnail requested could be saved.
NORAW_NOIMAGE_THUMB	12	Command	Neither raw or image requested could be saved.
NOTHING_SAVED	13	Command	None of the captured images could be saved.
NO_IMAGES_EXIST	14	Command	The list of images is empty; no images are yet saved in FLASH.
NO_SUCH_IMAGE	15	Command	The *img_id* did not match any stored image.
NO_SUCH_CHUNK	16	Command	The *chunknum* was not contained in the image, but the image existed.
DEFAULT_ERROR	17	Command	Default settings could not be loaded into current settings.
COMC_ERROR	18		Something critical has gone wrong, or a command sent did not follow the communication specifications.
UNKNOWN_COMMAND	19		A command was sent that followed protocol, but was not recognized by the camera system.

Table A.10: *Description of the status messages that the system can return. When a status message is returned from a command execution, the command number is returned after the status message. All status that are not DONE are preset with ERROR_.*

Arguments	Description
DATA	Registers and settings in a string; 8 bit register and 16 bit setting.
length	Length of the data string.
header	Each image has a corresponding header with info about the image; this is *img_id*,*time*,*size*,*type*.
time	The OBC time must be transferred as an argument to capture image.
thumb	This is a bit in one byte containing both thumb and raw. If the bit is set a thumbnail is generated.
raw	This is a bit in one byte containing both thumb and raw. If the bit is set raw image is saved.
img_id	The unique identifier for an image.
size	The size of an image.
type	What type of format is the requested *img_id*.
chunknum	The number of the chunk requested; the first chunk is number 0. Thumbs only have chunk number 0.
Command	When a status message is returned, it will contain the command number.

Table A.11: *Description of the arguments that can be used for some of the transferred commands. To see which commands takes arguments see* Table A.8.

HSN setup on CAM	Defined value
HSN_ADDRESS_CAM	6
HSN_PORT_CAM_CDH_CMD	0x60
HSN_PORT_CAM_CDH_SETUP	0x61
HSN_PORT_CAM_CDH_PING	0x11
HSN_BUF_HSN_PORT_CAM_CDH_CMD_LENGTH	6
HSN_BUF_HSN_PORT_CAM_CDH_SETUP_LENGTH	104
HSN_BUF_HSN_PORT_CAM_CDH_PING_LENGTH	2
HSN_PENDING_LIST_HSN_PORT_CAM_CDH_CMD_LENGTH	10
HSN_PENDING_LIST_HSN_PORT_CAM_CDH_SETUP_LENGTH	3
HSN_PENDING_LIST_HSN_PORT_CAM_CDH_PING_LENGTH	10
HSN setup on CDH	**Defined value**
HSN_ADDRESS_CDH	2
HSN_PORT_CDH_CAM_STATUS	0x60
HSN_PORT_CDH_CAM_DATA	0x62
HSN_PORT_CDH_CAM_PING	0x11
HSN_BUF_HSN_PORT_CDH_CAM_STATUS_LENGTH	3
HSN_BUF_HSN_PORT_CDH_CAM_DATA_LENGTH	255
HSN_BUF_HSN_PORT_CDH_CAM_PING_LENGTH	25
HSN_PENDING_LIST_HSN_PORT_CDH_CAM_STATUS_LENGTH	10
HSN_PENDING_LIST_HSN_PORT_CDH_CAM_DATA_LENGTH	10
HSN_PENDING_LIST_HSN_PORT_CDH_CAM_PING_LENGTH	10

Table A.12: *HSN port and address setup.*

Notes to User

At any given time a command can be transmitted to the camera system using the HSN protocol on the CAN interface running at 125 kb/s.

To test if the camera system is running send a HSN_CMD_CAM_CDH_PING over HSN on CAN and await reply. If a reply does not arrive within ten seconds, the camera system should be powered off and on.

The HSN protocol will receive an acknowledge, if the camera system has received a command. When the HSN transmitter has received the acknowledge, no further transmissions are sent from the camera system unless an error occurs or a command has been executed.

A.10.2 Jumper Settings

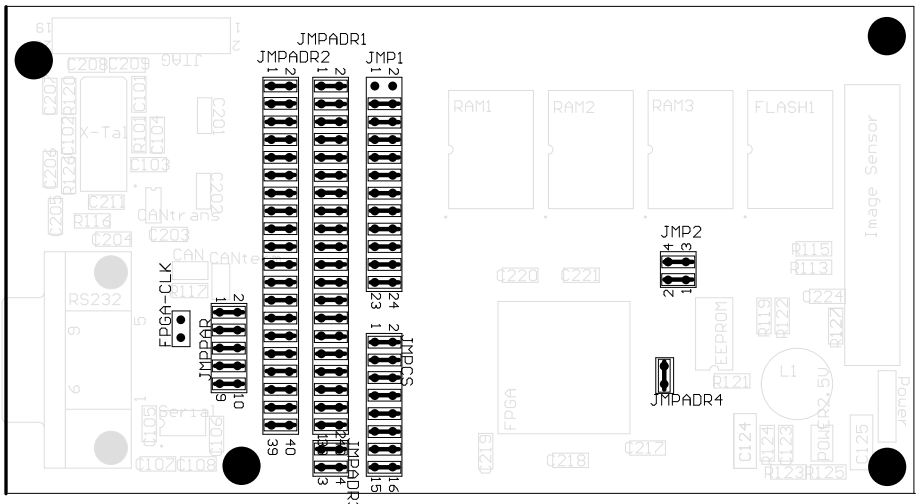


Figure A.17: *Jumper settings for camera system when fully operational.*

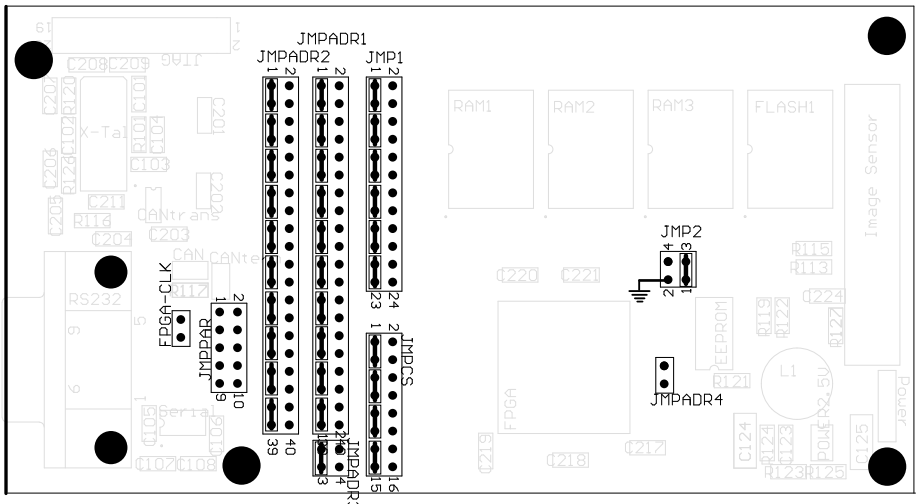


Figure A.18: *Jumper settings for camera system when running in minimum system mode. The GND symbol indicates that ground must be connected on this pin, also note that the minimum system configuration draws more power, as the FPGA is not configured.*

Index

Ørsted Satellite, 16

AAU Cubesat, 16, 32
AAUSAT-II, 6, 16, 32, 38, 41, 53, 60, 76, 90, 141
    OBC, 7, 52, 140, 154
AAUSAT-III, 6, 16, 19, 32, 41, 54, 80, 90, 135
ACS, 16
ADCS, 18
ARM, 33, 51, 55, 60, 63, 76, 77, 84, 91, 105, 110, 118, 149, 156, 164
    Advanced Memory Setup, 78
    EBI, 77, 78, 159
    PIO, 57, 61, 62, 79, 91
    Pipeline, 53, 94
    PLL, 59, 80, 145
    Reboot Mode, 77
    Remap Mode, 77
    RISC, 52
    Wait States, 67, 79, 159
ASIC, 72, 137
Assembly, 93

Beacon, 90, 165
Buck Converter, 54, 58, 122

C,C++, 77, 154
Cal Poly, 41
    PPOD, 16, 20
Camera
    Combined Lenses, 25
    Exposure Time, 27, 30
    Image Sensor, *see* Image Sensor
    ISO Speeds, 27
    Lens, 20, 23, 126
        Aperture, 26
        Focal Point, 24
        Light Beams, 24
        Magnification, 24
        Refraction, 24
    Shutter, 27
    Subject, 17
CAN, 32, 38, 42, 51, 58, 76, 86, 118, 122, 126, 140, 141, 164
    Data Link Layer, 141
    Dominant, 143
    Identifier Field, 141
    NRZ, 142
    Physical Layer, 141
    Recessive, 143
CCD, 19, 28
    Charge Transfer, 30
    Photodiode, 28
CDH, 17, 32, 35, 45, 76, 83, 85, 87, 94, 154
CMOS, 19, 29, 48
    Photodiodes, 29

Data Protection, 20
    Chunk, 22, 37, 43, 46, 83, 89, 120
    Hamming Coding, 21, 128
Devitech ApS, 7, 49, 50, 67, 97, 126, 164
    BlackEye, 7, 50, 127
        Temperature Sensor, 88, 91
DMA, 34, 50, 51, 55, 63, 76, 84, 93, 116
    Read-out, 57

eCos, 76, 77, 80, 84, 140, 154
    Booting, 85
    Interrupt Handling, 155
    MUTEX, 158
    Scheduler, 76, 84, 94, 154
        Bitmap, 155
        Lottery, 155
        Multi Level Queue, 155
    Semaphore, 85, 157
    Setup, 80
    Threads, 154
EEPROM, 58, 72
EMC, 60, 150
    Common-mode Choke, 151
    Critical Length, 60
    Crosstalk, 60, 152
    Decoupling Capacitors, 60, 150
    Ferrite Bead, 151
    Ground Bus Noise, 60, 150
    Power Bus Noise, 60, 150
    Transients, 74, 150
    Transmission Line Reflections, 60, 150, 151
EPS, 35, 54, 75
ESA, 7, 41, 60

FLASH, 36, 53, 56, 60, 67, 79, 80, 87, 91, 105, 118, 122, 123, 147, 159, 162
FLP, 35
FPGA, 7, 34, 53, 55, 60, 62, 76, 79, 91, 106, 122, 147, 159, 164
    CLB, 62, 74
    IOB, 62
    One-hot Encoding, 65
    Time Constraints, 69
    Timing Diagrams, 161

VHDL, *see* VHDL

GND, 17, 18, 35
GPV Printca A/S, 7, 60, 127

HSN, 32, 38, 42, 45, 58, 76, 86, 88, 122, 126, 142, 164
Destination, 38
Frame Type Field, 38
Frame-Ack, 39
Interface, 164
Ports, 38
Source, 38
Timeout, 39

I^2C, 32
Image Formats
Raw, 17, 50, 55, 65, 83, 88, 94, 127
RGB, 49, 94, 98, 123, 127
Thumbnails, 36, 43, 46, 53, 76, 83, 88, 98, 117, 123
YCbCr, 99, 117
Image Handling
Compression, 88, 98, 123
AIC, 103
Bit Stream, 98
Block Prediction, 101
Color Space, 99
DCT, 101
Entropy, 98, 101
Lossless, 98
Lossy, 98
Interpolation, 88, 94
Bayer Filter, 94
Bilinear, 95
CFA, 94
Constant Hue, 96
Demosaicing, 95
Gradient Based, 97
Resizing, 98
Thumbnail, 88, 89
Image Sensor, 28, 32, 49, 55, 63, 76, 95
CCD, *see* CCD
Charge Collection Efficiency, 30
CMOS, *see* CMOS
Electronic Shutter, 30
Initialization, 88, 91
Quantum Efficiency, 30, 95
Rolling Shutter, 31
Serial Protocol, 92
Settings, 87
Skip, 50
Snapshot Mode, 162
Windowing, 50
INSANE, 38

JTAG, 57, 60, 126, 164

LEO, 16, 19, 20
LQFP, 52

MCC, 17, 35, 83, 85, 89
MECH, 20, 48, 54, 127
Methods
Object Orientated Design, 139
Remote Programming, 140
SSU, 135
UML, 135, *see* UML
Users Manual, 135
Microcomputer
CPU, *see* ARM
FLASH, *see* FLASH
FPGA, *see* FPGA
Harvard Structure, 32
PEEL, *see* PEEL
RAM, *see* RAM
Von Neumann Structure, 33, 51
Microcontroller, 51
ARM, *see* ARM
MIPS, 51
Motorola 68000, 51
RISC, 52

OBC, 17, 32, 50, 80

PCB, 50, 59, 77, 122, 126, 150, 163
EMC, *see* EMC
FR4, 127
Glass Polyimide, 60
GND & V_{CC} Planes, 60
PEEL, 34, 53
Pico Satellite, 6, 16
PPOD, *see* Cal Poly

Radiation effects
Displacement Damage, 19
Single Event Burnout, 19
Single Event Latchup, 19
Single Event Upset, 19
Total Ionizing Dose, 19
Radiation in Space
Packaging, 20, 49
Radiation Hardened, 20, 48
Software Protection, *see* Data Protection
South Atlantic Anomaly, 19
Van Allen Belts, 19
RAM, 53, 60, 88, 147, 159
Requirements
Functional Requirements, 42
Satellite Requirements, 41
RS-232, 57, 58, 60, 126, 140, 164

SMD, 7, 60
BGA, 49, 127
LQFP, 52
Software

HSNTest, 122, 140
Modules, 75
Pass-through Mode, 64, 73, 93
Processes, 75
Capture Image, 55
Image Handling, 76
Read-out Sequence, 55, 62, 64, 66, 73, 94
SRAM Software, 84, 85, 88, 93
Threads, 84
ComC_thread, 76, 85
Debug_thread, 140
MAIN_thread, 76, 85, 88
Use Cases
Capture Image, 36, 76
Delete Image, 37
List Images, 36
Send Image, 37
Setup Camera, 35
Solithane 113300, 20, 49
Spacelink, 17, 87, 98
SSETI Express, 16

UML
Active State Charts, 84
Use Cases, 34, *see* Software

VHDL, 7, 71
Designing, 72
Modules, 71
Programming, 72
Synthesis, 71
Synthesizeable, 71

B Flowcharts

B.1 Description of the UML Active State Chart Figures Used in Flowcharts

The definitions on how to read the UML active state chart figures used in the following flowcharts are here described to avoid misinterpretations of the charts.

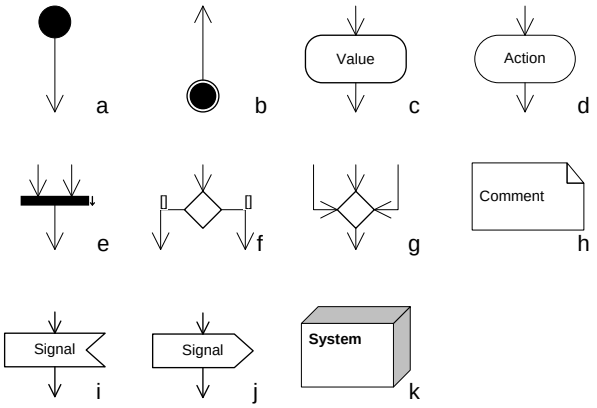


Figure B.1: *The figures used in the flowcharts.*

On Figure B.1 the used figures are seen, they are all described below.

a A black dot is used to represent initializing, this is always the starting point of the chart.

b A block dot with a surrounding circle is used to represent an end of flow. This results in the function described in the chart to return to the calling function.

c A rounded rectangle is used to inform what the return value should be.

d An elongated circle is used to describe an action that should be performed.

e A black thick line is a waiting state; only when all the actions pointing to the waiting state are done the output is followed.

f A tilted diamond gives two options; in this case it is a decision. The arguments on the lines going out defines when to take that route. These are often expressed as boolean.

g The other configuration of the tilted diamond. Incomming paths are all redirected to follow the output.

h A comment box, it is an opportunity for the designer to give further explanations than the chart allowed for.

i Incomming signal from other components.

j Outgoing signal to other components.

k A component or a system.

Figures are interpreted from UML 2.0 [OMG, 2004, 346].

B.2 Boot Sequence

Initialized by EPS.

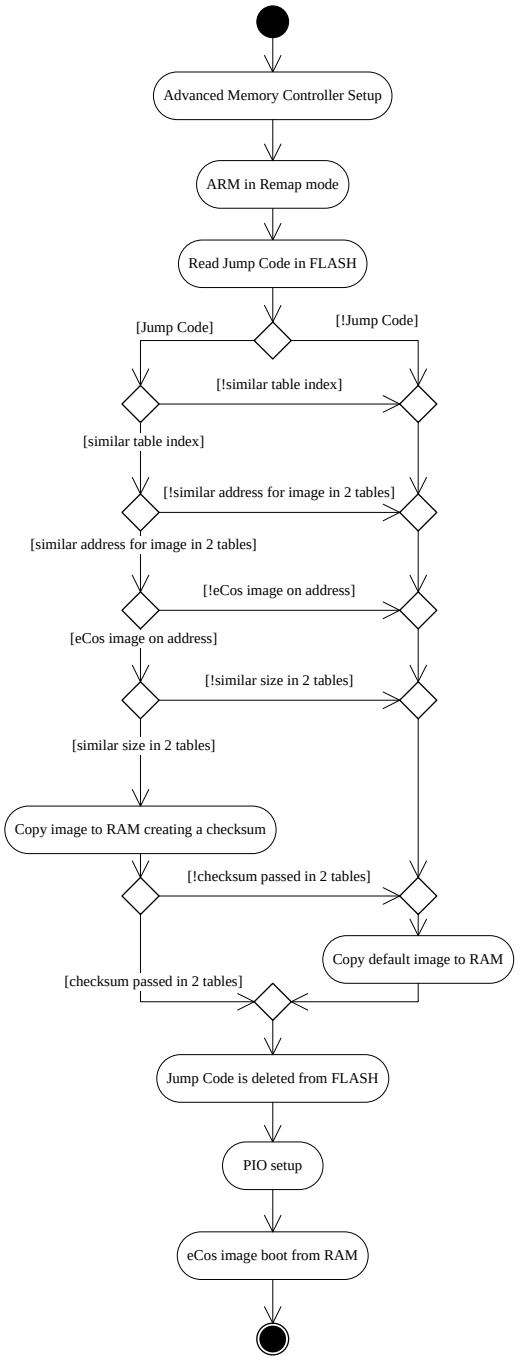

Advanced Memory Controller Setup
ARM in Remap mode
Read Jump Code in FLASH
[Jump Code]
[!Jump Code]
[!similar table index]
[similar table index]
[!similar address for image in 2 tables]
[similar address for image in 2 tables]
[!eCos image on address]
[eCos image on address]
[!similar size in 2 tables]
[similar size in 2 tables]
Copy image to RAM creating a checksum
[!checksum passed in 2 tables]
Copy default image to RAM
[checksum passed in 2 tables]
Jump Code is deleted from FLASH
PIO setup
eCos image boot from RAM

Figure B.2: *The booting of a new eCos image is verified, else Default is booted.*

B.3 MAIN_thread

Initialized by bootcode.

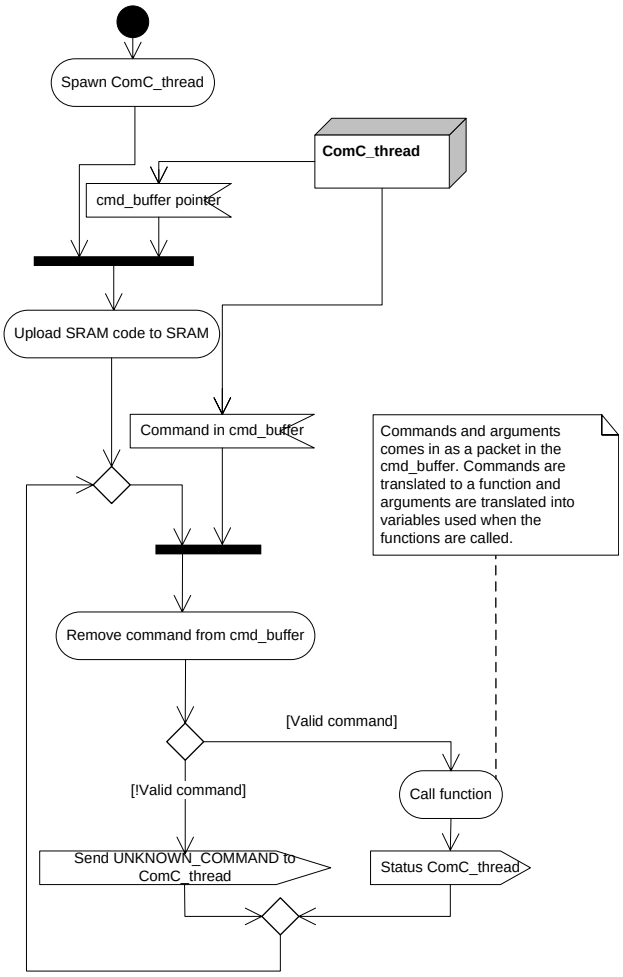

Spawn ComC_thread
ComC_thread
cmd_buffer pointer
Upload SRAM code to SRAM
Command in cmd_buffer
Commands and arguments comes in as a packet in the cmd_buffer. Commands are translated to a function and arguments are translated into variables used when the functions are called.
Remove command from cmd_buffer
[Valid command]
[!Valid command]
Call function
Send UNKNOWN_COMMAND to ComC_thread
Status ComC_thread

Figure B.3: *The image handling thread illustrated as a flowchart spawns a communication thread and runs one function at a time if a command is present in the command buffer.*

B.4 ComC_thread

Spawned by MAIN_thread to handle communications with CDH.

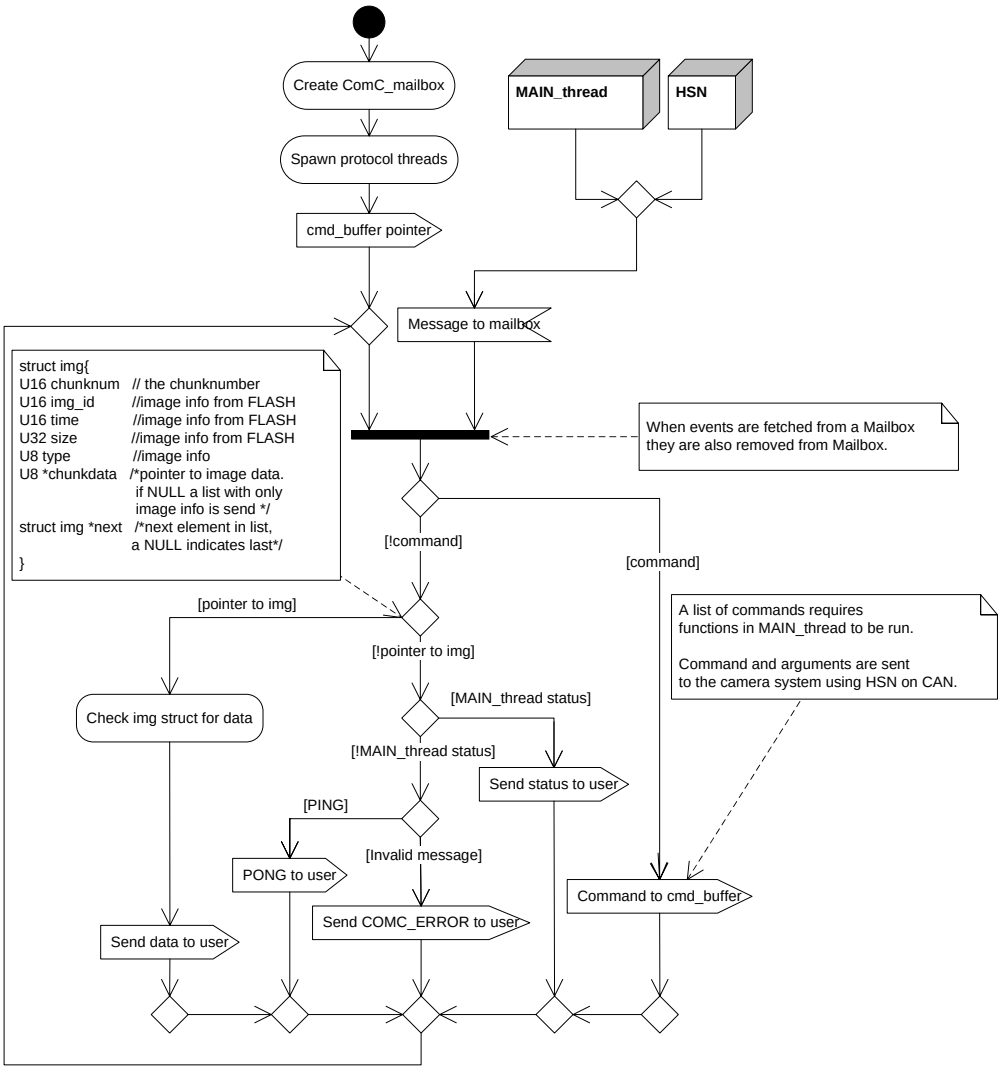


Figure B.4: *The communication thread illustrated as a flowchart is checking for incoming commands and if one is detected it adds the command to the command buffer.*

B.5 int cam_setup(void *data, U8 length);

Registers and settings are transferred as a string of data. The string contains 8 bit and 16 bit fields indicating register and settings.

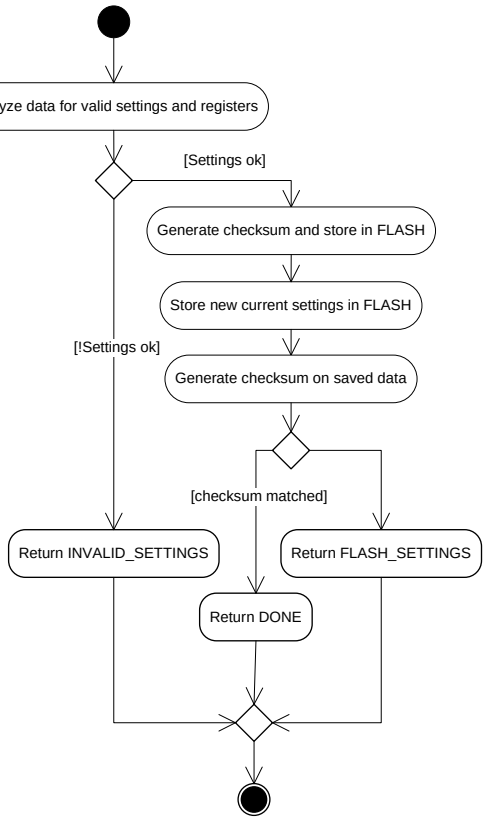


Figure B.5: *The module setup cam is implemented as a function in MAIN_thread.*

B.6 int cam_defaults();

This function is called to set the current used image sensor settings to default.

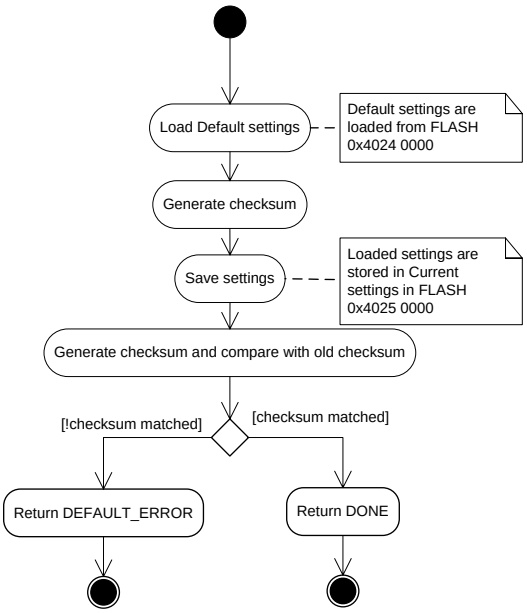

Load Default settings
Default settings are loaded from FLASH 0x4024 0000
Generate checksum
Save settings
Loaded settings are stored in Current settings in FLASH 0x4025 0000
Generate checksum and compare with old checksum
[!checksum matched]
[checksum matched]
Return DEFAULT_ERROR
Return DONE

Figure B.6: *The module default settings is implemented as a function in MAIN_thread.*

B.7 int cam_capture(int time, bol thumb, bol raw)

time specifies the time that capture was requested. *thumb* and *raw* are True/False indicators.

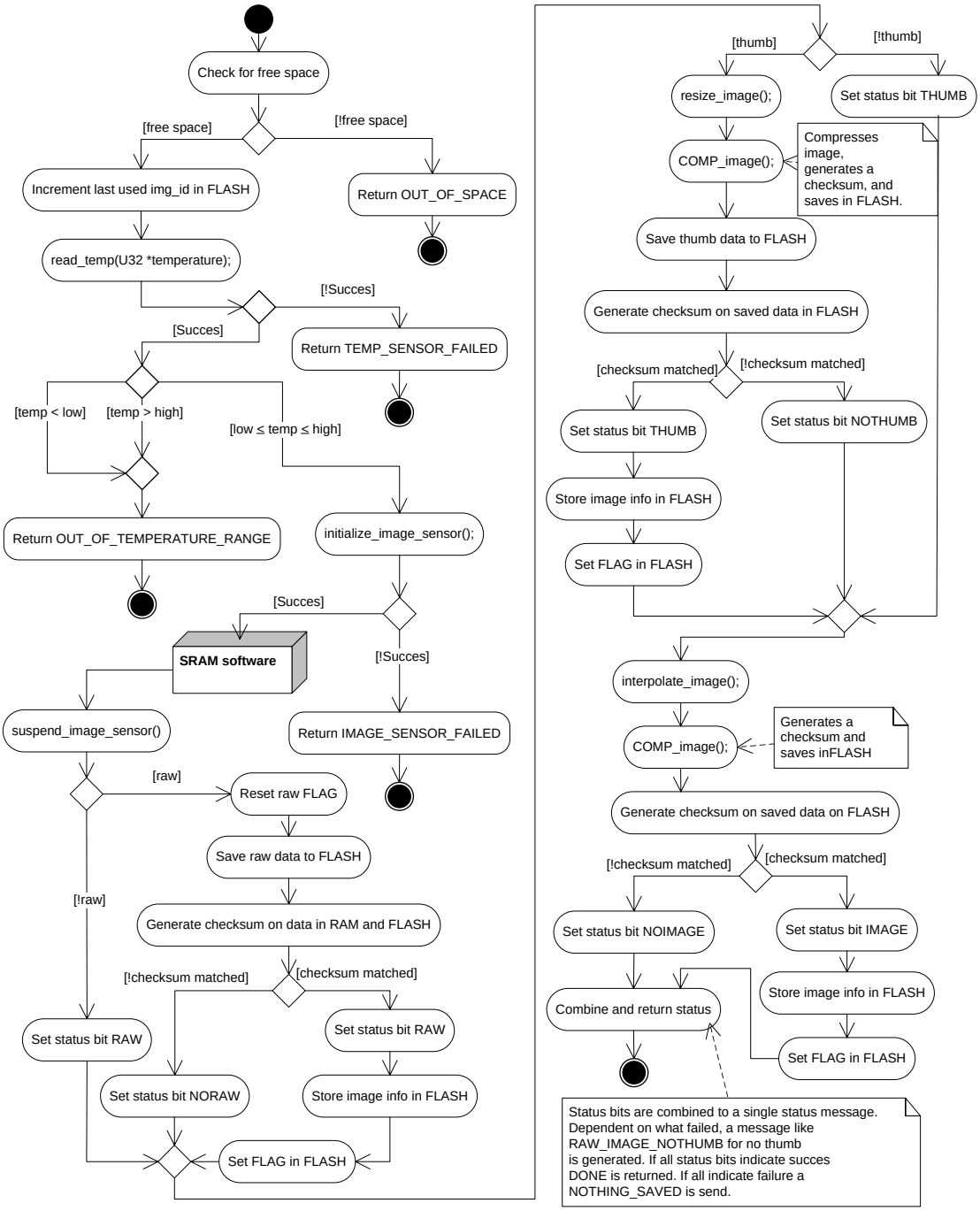


Figure B.7: *The module capture image is implemented as a function in MAIN_thread.*

B.8 int cam_list_images();

A pointer to a linked list of structs of the type img is sent to ComC_thread. They contain information about the saved images, thumbs, and the raw image to ComC_thread. However, the actual image data pointer is set to NULL, so that ComC_thread does not send the image data.

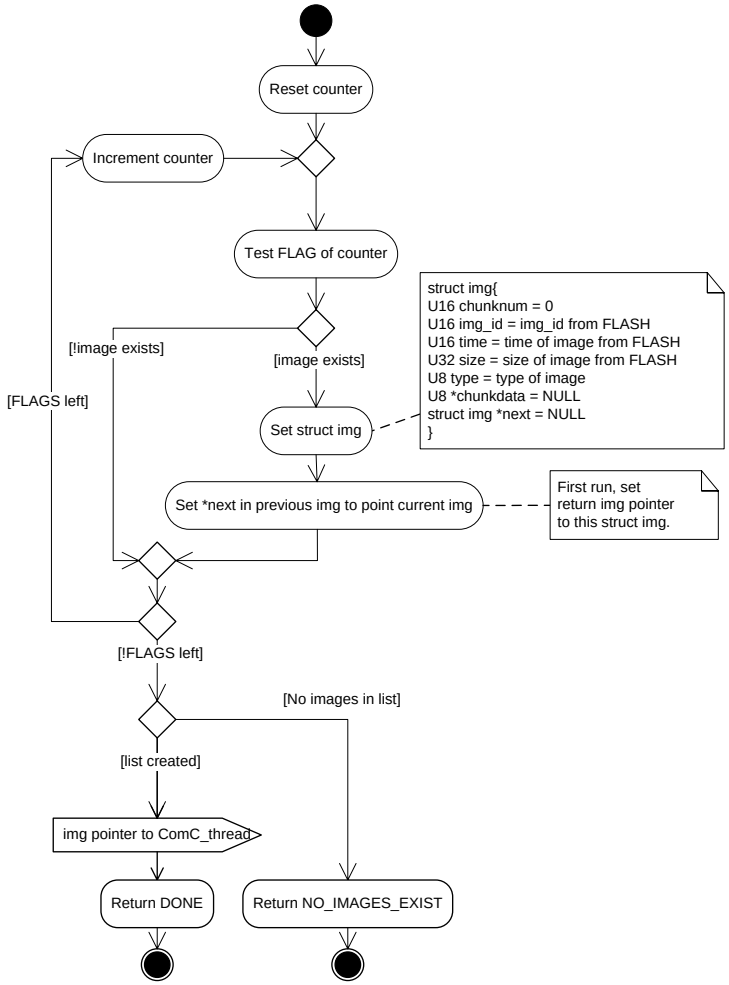


Figure B.8: *The module list images is implemented as a function in MAIN_thread.*

B.9 int cam_delete_image(int img_id);

The specified image is removed if it exists. The corresponding thumbnail is also removed.

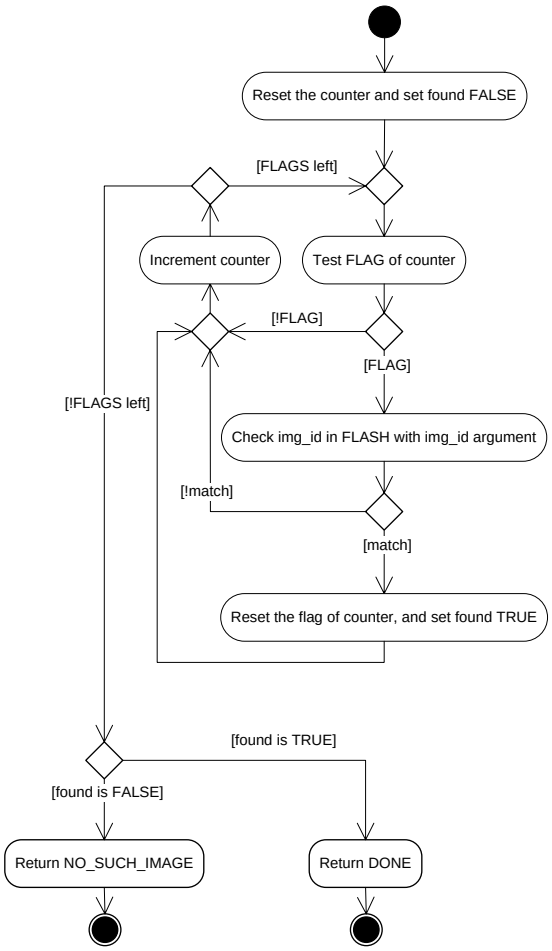


Figure B.9: *The module delete image is implemented as a function in MAIN_thread.*

B.10 int cam_send_img(int img_id, int chunknum, U8 type)

Sends either a chunk in a struct of the type img or a list of chunks. A pointer to this type is send to ComC_thread that resends data of the struct to CDH. The next chunk pointer can be set to NULL if a single struct of data is returned; else it points to a structure of type img to send when this one is received by CDH.

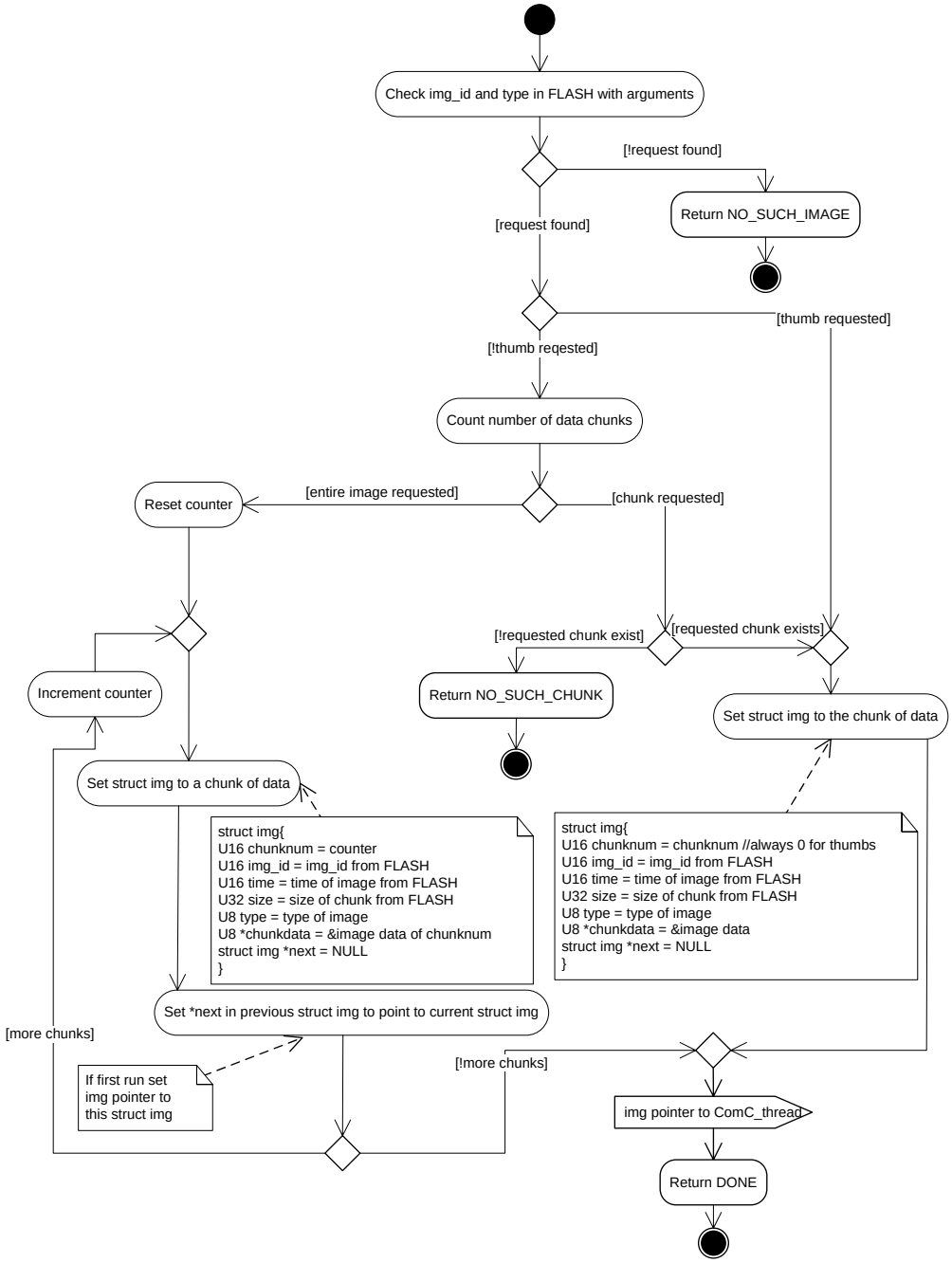

Check img_id and type in FLASH with arguments
[!request found]
Return NO_SUCH_IMAGE
[request found]
[thumb requested]
[!thumb reqested]
Count number of data chunks
Reset counter
[entire image requested]
[chunk requested]
[!requested chunk exist]
[requested chunk exists]
Increment counter
Return NO_SUCH_CHUNK
Set struct img to the chunk of data
Set struct img to a chunk of data
struct img{
U16 chunknum = counter
U16 img_id = img_id from FLASH
U16 time = time of image from FLASH
U32 size = size of chunk from FLASH
U8 type = type of image
U8 *chunkdata = &image data of chunknum
struct img *next = NULL
}
struct img{
U16 chunknum = chunknum //always 0 for thumbs
U16 img_id = img_id from FLASH
U16 time = time of image from FLASH
U32 size = size of chunk from FLASH
U8 type = type of image
U8 *chunkdata = &image data
struct img *next = NULL
}
Set *next in previous struct img to point to current struct img
[more chunks]
If first run set
img pointer to
this struct img
[!more chunks]
img pointer to ComC_thread
Return DONE

Figure B.10: *The module send image is implemented as a function in MAIN_thread.*

B.11 int cam_housekeeping(U16 *hk)

Puts a string of data in RAM starting at the address of *hk*. The length of the string is specified elsewhere.

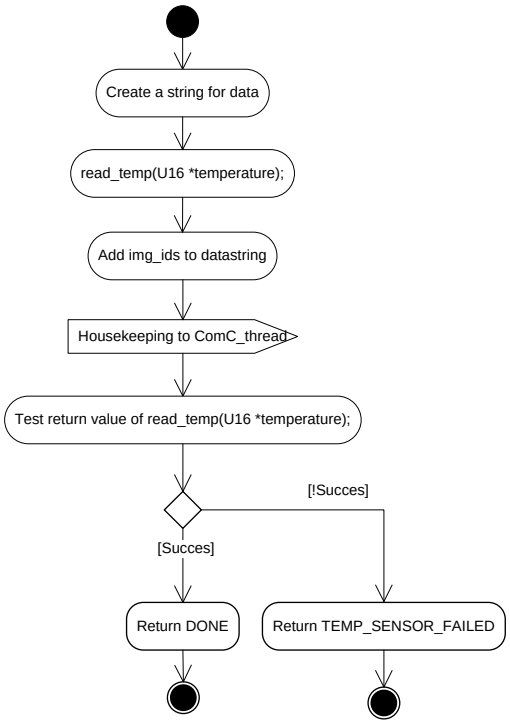


Figure B.11: *The module housekeeping is implemented as a function in thread MAIN_thread.*

C Schematics

D Attached CD

D.1 Contents of CD

- Report in digital format
- Bibliography
- Datasheets
- Debouncing Circuitry
- Image Compression (AIC - Linux software, AIC - The original source code, JPEG - The original source code, and Sample pictures)
- Schematics (Both pdf and Altium Designer files)
- Source Code for ARM
- doxygen documentation
- Source Code for FPGA
- Source Code for HSNTest